Technische Universität München
TUM School of Computation, Information and Technology

Advancing Privacy-Enhancing Technologies for Policy-Driven Data Sovereignty and Provenance

Jan Philipp Lauinger

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz: Prof. Dr.-Ing. Antonia Wachter-Zeh

Prüfende der Dissertation:

1. Prof. Dr. Sebastian Steinhorst
2. Prof. Dr. Arthur Gervais

Die Dissertation wurde am 25.09.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 24.02.2025 angenommen.

Acknowledgments

I would like to thank my supervisor Prof. Dr. Sebastian Steinhorst for the opportunity to work in a very interesting field of research. Thank You for keeping me motivated, for providing valuable guidance, and for discussing any concerns at any time throughout this project. Additionally, I would like to thank my second examiner, Prof. Dr. Arthur Gervais.

I want to thank my team at the Associate Professorship of Embedded Systems and the Internet of Things for all discussions and the kind atmosphere during the past years. In addition, I would like to thank all my coauthors for spending time and energy on our papers.

I express special thanks to my colleague Jens Ernstberger. I see our endless discussions, meetings, and updates as the core driver of the research we conducted. This work would not have been possible without You.

Last but not least, I would like to thank my family and girlfriend for supporting me throughout this journey.

Abstract

Digital data is a key driver to improve services on the Internet and, with that, navigate commercial success. The adoption of new online services and devices led to an increase in digital data and transformed the potential of the online economy. Until data leaks, unaware analytics, accurate marketing, and the violation of human rights raised concerns about the administration of digital data. Today, governments regulate the sovereignty of digital data and impose new obligations on digital services to empower consumers or end devices. For many of the new requirements, the question remains if and how new laws can be realized. Besides the trend of new legislation, emerging open infrastructures explored and re-evaluated the autonomy of digital data using modern cryptography. In this context, Privacy-enhancing Technologies (PETs), which deliver key principles to balance the protection and transparency of digital data, accelerated a transformation towards a decentralized and responsible data economy. But, as of today, open infrastructures and consequent innovations remain isolated and have not been considered for the improvement of existing protocols.

In this thesis, we investigate the potential of PETs in the context of data sovereignty. Based on our findings, we seek to improve the de facto standard of online protocols towards a policy-driven and privacy-aware version. Our goal is to provide efficient building blocks for data sovereignty and provenance which solve upcoming dilemmas of compliance, liability, and protection. As a result, our contributions improve the status quo as follows. Our first contribution secures the self-determined protection of digital data and accounts in public infrastructures. In addition, we present a privacy-preserving authentication scheme and use it to create accountable online interactions. These solutions explore new approaches to data compliance where nothing beyond self-determined facts on digital data is disclosed. Our next contributions target data provenance protocols which return ownership of custodial data back to end devices. We optimize the efficiency of privacy-preserving protocols for the verification of data provenance and facilitate client-side deployments in constrained environments. Here, our first optimization improves bandwidth requirements by reassessing the usage of PETs in a setting with a weaker network adversary. Additionally, we show that, in an asymmetric privacy setting, honest-verifier cryptographic proof systems can be secured against malicious adversaries. As a result, we obtain highly efficient execution times of cryptographic proof computations. Our third contribution automates the compilation of user-driven policies to PET computation circuits. With that, we mitigate the liability of custodial data controllers to define, enforce, and maintain data policies on behalf of users. The contributions provided in this thesis serve as core building blocks to build data-sovereign and policy-compliant applications of the future Internet.

Kurzfassung

Digitale Daten sind ein zentraler Treiber für die Verbesserung von Internetdiensten und sorgen damit für kommerzielle und private Erfolge. Die Einführung neuer Online-Dienste und -Geräte führte zu einer Zunahme digitaler Daten und veränderte das Potenzial der Online-Wirtschaft. Allerdings ließen die Ereignisse in Form von Datenlecks, unbewusster Datenanalysen, datenbasiertes Marketing und die Verletzung von Menschenrechten Bedenken hinsichtlich der Verwaltung digitaler Daten aufkommen. Heutzutage regulieren Regierungen die Souveränität digitaler Daten und stellen digitale Dienste vor neue Verpflichtungen, welche dem Einfluß von Verbrauchern und Endgeräten auf digitale Daten zugute kommt. Bei vielen der neuen Anforderungen stellt sich die Frage, ob und wie neue Gesetze effizient und maßstabsgetreu umgesetzt werden können. Darüber hinaus zeigen offene Infrastrukturen im Zusammenspiel mithilfe moderner Kryptographie eine neue Autonomie digitaler Daten. In diesem Zusammenhang beschleunigen datenschutzfördernde Technologien den Wandel hin zu einer dezentralen und verantwortungsvollen Datenwirtschaft. Bis heute bleiben jedoch offene Infrastrukturen und die daraus resultierenden Innovationen im Bereich von datenschutzfördernden Technologien isoliert und weitestgehend ungenutzt.

In dieser Dissertation wird das Potenzial von datenschutzfördernde Technologien im Kontext der Datensouveränität untersucht. Basierend auf den Erkenntnissen verbessern wir Online-Protokolle hin zu einer richtliniengesteuerten und datenschutzbewussten Umsetzung. Unser Ziel ist es, effiziente Bausteine für Datensouveränität und Datenherkunft bereitzustellen, die bevorstehende Dilemmata von Gesetzeseinhaltung, Haftung und Datenschutz lösen. Die Ergebnisse unserer Forschung verbessern den Stand der Technik wie folgt: Unser erster Beitrag sichert den selbstbestimmten Schutz digitaler Daten und Konten in öffentlichen Infrastrukturen. Darüber hinaus stellen wir ein datenschutzschonendes Authentifizierungsschema für die Erstellung verantwortungsvoller Online-Interaktionen vor. Diese Lösungen erforschen neue Ansätze der Datenkonformität, bei denen nichts außer selbstbestimmte Fakten über digitalen Daten offengelegt werden. Unsere nächsten Beiträge untersuchen und verbessern Datenherkunftsprotokolle, welche das Eigentum und die Kontrolle von geteilten Daten an Endgeräte zurückgeben. Wir optimieren die Effizienz von Datenherkunftsprotokollen unter Einhaltung datenschutzrechtlicher Gesetze und erleichtern die Bereitstellung von Datenherkunftsprotokollen in eingeschränkten digitalen Umgebungen. Unser letzter Beitrag automatisiert die Kompilierung von benutzergesteuerten Richtlinien für den Einsatz von datenschutzfördernden Technologien. Damit wird die Verantwortung einer gesetzesgetreuen Datenverarbeitung nach benutzerdefinierten Richtlinien an Technologie gebunden.

Contents

1. Introduction

In this thesis, we investigate new techniques to improve the data consent-protection dilemma of modern online services and devices. Our main contributions improve the efficiency and interplay between privacy-enhancing technologies and open infrastructures and provide operating bodies with transparent, verifiable data protection and compliance techniques.

We argue that consent and protection concerning new safeguarding, liability, and regulatory requirements of digital data is only possible if online entities can reclaim control and authority over their digital data. We believe that control and authority over data are tied to empowering technologies that reside in the hands of the entities that are affected by new regulations.

The following sections motivate the context of this thesis, introduce the research questions, and present an overview of our main contributions.

1.1. Motivation

The Internet of today is at a turning point, where legislation is likely to shape the future of the Internet ecosystem as never before. The reason why governments concentrate on releasing legislative policies depends on growing concerns and past events that impact the competitiveness of individual economies. At the same time, innovations in the field of public, decentralized infrastructures yield a plethora of privacy-empowering technologies that could help established Internet protocols achieve compliance with new policies.

Digital Legislation: Looking into the past and the early days of the Internet, online data was collected, managed, and controlled by separate Internet services individually. However, the increase of services accessible through the Internet introduced usability issues if users or devices manually handled accounts for an ever-growing number of services. The advent of federated identity mitigated the growing need for a single account with compatibility across multiple services. Federated identity relies on a trusted third party that delegates authentication, access, and authority on behalf of users or devices. For example, the Single Sign-on (SSO) paradigm enables users to select third-party identity providers (e.g. Google, Meta, etc.) to authenticate towards multiple online services with a single account. Thereby, identity providers maintain the accounts of registered users or devices. The concept of federated identity quickly led to other problems.

Protocols, which delegate the authority of users, empower intermediaries (e.g. identity providers) to decide which data to track, use, or share with other online services. Additionally, based on the account activity, intermediaries may accumulate and analyze account data for different economic purposes. From an internal perspective, account data is used to enhance owned products and, with that, create better customer experiences. From an external view, account behavior is of interest to companies that require improved business intelligence (marketing, planning, advertisement, etc.). Further, the initial lack of transparency in the intermediary's terms and conditions caused privacy concerns about how user data was used and shared. Neither was is possible to determine if and how intermediaries took economic advantage of the position as the intermediary.

The evolution of the SSO protocol yielded an online economy where few centralized identity providers succeeded. The situation of centralized providers with accumulated data attracted attackers and caused data breaches of new scales [1]. The new scales of data breaches triggered a collective awareness of how valuable data can be and how little influence users have about circumventing existing arrangements (e.g. opting out of tracking or data sharing agreements towards a third party). The data leaks confirmed emerging privacy concerns until noticing that identity providers violated the fundamental human right to be forgotten [2]. The violation of law provoked legislative engagement of the United States (US) and European Union (EU) where the California Consumer Privacy Act (CCPA) or the General Data Protection Regulation (GDPR) handled data privacy and protection regulation constitutionally [3], [4]. These regulations imposed new rulings on online services as well as intermediaries. From here on, data controllers and processors had to consider and implement user-centric authority, accountability, and data sovereignty that includes a user's rights to know, delete, opt out, non-discriminate, limit, and correct data.

As a continuation of data protection regulations, the EU adopted adequacy for the EU-US Data Policy Framework (DPF) to safeguard national interests during cross-border data transfers [5]. DPFs (e.g. EU-US DPF, Swiss-US DPF, etc.) ensure that national data protection laws sufficiently apply in partnering economic areas. With that, digital information transfers and data privacy move towards cross-border arrangements that we similarly experience at borders in our physical world. As another legislative policy, the EU Digital Services Act (DSA) came into force on February 17th, 2024. The DSA is a legal framework with the purpose of safeguarding legal certainty for businesses operating online. As such, the DSA fights disinformation, and illegal content and enforces transparent advertising. The enhanced regulatory obligations of the DSA affect digital platforms or content disseminating users or devices. On one hand, the DSA lowers liability concerns of EU-based services that support the fundamental functioning of the Internet. On the other hand, the DSA categorizes online services according to their size and purpose and defines new transparency and due diligence obligations [6]. Thus, in the current state of the Internet, the ever-increasing regulatory complexity will continue to challenge online services and end devices.

The consequences of non-compliance are penalty fines of up to 4% of worldwide annual revenue [7]. Thus, online services are faced with increased expenses to investigate and implement new regulatory obligations. To solve upcoming liability and compliance requirements, computer research is challenged to come up with alternative approaches to data administration. A promising direction is the idea to alleviate data controllers through the equipment of data sovereign principles at end devices. With that, instead of data controllers, end devices could be held accountable for digital interactions. Another interesting idea is to offload data responsibilities towards emerging public digital infrastructures.

Public Digital Infrastructures and Privacy-enhancing Technologies: Besides the increase of digital laws and policies, another recent Internet evolution took place. With the advent of decentralized public infrastructures in the form of blockchains, ownership of digital assets and computation has been put back into the hands of users. Blockchains equip end devices with wallets that contain cryptographic key pairs. In this context, cryptographic keys are used to protect data integrity in the form of signatures, which are issued by the entity controlling the wallet. In blockchain ecosystems, it holds that every interaction is authenticated via a wallet signature. For instance, if a device issues a signature to invoke blockchain functionalities, then the device remains accountable for the interaction within the blockchain system. Thus, if the wallet owner maintains cryptographic keys in a non-custodial model, wallets grant data sovereignty and offer opposition to current online protocols that take over the data administration on behalf of users.

In contrast to the above-discussed centralization that exists in the Internet today, blockchains operate in a decentralized setting. Blockchain operators are decentralized and run a suite of blockchain-specific protocols and algorithms on proprietary or rented computers. Since blockchains count as public infrastructures, anyone can join and participate as a node operator. The core part of blockchains is a consensus protocol that maintains a secure state of data blocks. The consensus protocol ensures that every operator validates the correctness of new state updates before locally applying the state update. As such, blockchain operators come to a network-wide agreement of a distributed but equal view of data blocks. Even though blockchains are decentralized and users can interact with many operators, privacy violations (e.g. tracking opportunities) must be reassessed. Despite ongoing research challenges, the decentralized protocol design and shared authority between blockchain operators have the potential to optimize existing online protocols. Further, public digital infrastructures could be used to build alternative Internet protocols with user-driven privacy policies.

Due to the fact that cryptography is fundamentally baked into blockchain protocols (every state update is cryptographically verified at every operator), blockchain research has yielded many innovations in the field of applied cryptography. For example, the number of state updates a blockchain network can take is handled by Privacy Enhancing Technology (PET) in the form of advanced cryptographic proof systems [8], [9]. Multi-party Computation (MPC) as another domain of PETs is used to investigate shared custody aspects of wallet management to enhance the reliability of wallet management [10]. A combination of MPC techniques and cryptographic proofs is used in data provenance protocols, which secure blockchain state updates by attesting to the data integrity of secure channels [11], [12]. Using PETs to disguise public parts of state updates, and the state of blocks in general has been an early goal of blockchain research [13]. Hence, blockchain ecosystems will likely continue to accelerate innovations of applied cryptography, and, with that, serve as an interesting technology pool that empowers data sovereign protocols and applications.

Summary: In this thesis, we research applied cryptography in the context of building data-sovereign solutions. By advancing PETs, we aim to secure, extend, or optimize protocols with strong data sovereignty guarantees. The contributions of this thesis help build and deploy new forms of digital sovereignty and compliance.

1.2. Research Questions

In the following, we derive three core research questions from the situation discussed above. Before we present a concise representation of our research questions, we narrow down the context to three core challenges.

Data Consent - Protection Dilemma: Public digital infrastructures and PETs achieve alternative forms of digital data sovereignty. However, the question remains if and how online services compensate for new liability and compliance obligations. If device data is protected by service providers, then providers have the responsibility to define and collect consent from users. Here, devices hand over trust to providers that are assumed to protect data according to agreements. With new data protection and privacy obligations, service providers face new liabilities and costs in managing consent agreements transparently. On the other hand, relieving service providers from upcoming challenges by returning responsibilities of data administration back to devices removes consent agreements and gives devices the authority to protect data. In this setting, data sovereignty is tied to end devices that can introduce usability and reliability issues (e.g. lost wallet passphrases) [14].

An interesting direction of research is handing over parts of the consent and protection challenges to public digital infrastructures. The immutable and accountable management of consent agreements at public infrastructures removes extra costs and responsibilities from service providers. At the same time, the satisfaction of agreements and consent is publicly visible and verifiable if data remains in public infrastructures. Concerning data protection, privacy-critical data should never be stored immutably "on-chain" in public infrastructures. Otherwise, the policy of the "right to be forgotten" is violated. If data is maintained at end devices using data sovereign techniques, then PETs can ensure that the data administration "off-chain", away from public infrastructures, respects publicly exposed policies and laws. As a consequence, we formulate our first research question according to the challenge of guaranteeing self-determined data protection and decision-making for consent in public digital infrastructures. Data protection according to policy-driven agreements must hold for online accounts at the creation and throughout online interactions with other services. Notice that our challenge depends on the interplay of PETs and public digital infrastructures, which is an active research domain on its own. Ideally, the new solution mitigates the transparent exposure of account behavior to the public.

Limits of Privacy Enhancing Technologies: Beyond new solutions for digital accounts, this work investigates how ownership of custodial data can be brought back to end devices and locally maintained accounts. We focus on data provenance technologies, which return data ownership from custodians via data requests through secure communication channels. The core driver of data provenance protocols is PETs, which give devices the opportunity to prove the origin and, with that, ownership of data extracted from secure channels. PETs became prominent in the form of Zero-knowledge Proofs (ZKPs). Initially, ZKPs brought confidentiality for digital currencies [15] and secured digital credentials with new forms of privacy [16]. Thereby, researchers invented dedicated ZKP protocols to prove specific statements. Subsequently, general-purpose proof systems have been able to prove arbitrary statements expressed via programmable computation circuits [17]. Today, researching zero-knowledge Virtual Machines (zkVMs) and optimizing proving efficiency for any type of computational arithmetic count to the most researched ZKP topics [18].

The topics researched in this thesis not only rely on efficient ZKP protocols. For example, data provenance protocols, which empower devices to verify the provenance and ownership of data from secure channels, rely on ZKP technology and secure MPC techniques. MPC technology introduces additional overheads and limits an Internet-wide deployment of data provenance protocols [12]. Another challenge is the deployment of privacy-preserving data provenance protocols. Here, current ZKP approaches are a bottleneck and remain impractical in constrained environments (e.g. browsers, Internet of Things (IoT) devices, mobile phones). In order to create practical building blocks for the construction of data-sovereign solutions, the above defined challenges must be solved. Thus, we define our second research question based on the limits of PETs in data provenance protocols.

Verifiable Data Sovereignty Through User-driven Policies: With access to sovereign accounts and custodial data, end devices still depend on data sharing and external processing of data when participating in the digital economy. As a consequence, we investigate how data sovereignty can be defined over policies such that the device controls data sharing and external data processing. In the context of this work, consent to data access and processing should be transparently exposed on public digital infrastructures. Further, if processing applies "on-chain" in public infrastructures, then the public and devices themselves are able to track and comprehend outsourced processing and access. However, if the processing of shared data happens "off-chain", at individual commercial or private services, then devices cannot simply assess if the processing satisfied previous agreements. Further, devices have to maintain data sovereignty and remain empowered to determine the policy that guides external processing and sharing activities of data.

The above depicted problem is an active research area and one potential solution works as follows. Outsourced data processing in proprietary infrastructures can be made verifiable through PETs, where data processors attest to a compliant data processing by attaching a ZKP [19]. Here, punishing mechanisms ensure that processors lose valuable deposits if the ZKP provision does not happen. The complexity of the solution is difficult to solve in practice and opens many challenges. To contribute to the research field, our third research question investigates how PETs function in line with user-driven policies to mitigate unconscious sharing and processing of data.

In summary, we define the concise research questions of this thesis as follows:

- **Research Question 1:** How can we ensure self-determined protection and transparency of digital data during the creation and interaction of online accounts?
- **Research Question 2:** How can we practically return the sovereignty of custodial data back to devices and users?
- **Research Question 3:** How can we mitigate the unconscious sharing and processing of custodial data through user-driven policies?

1.3. Organization

The structure of the thesis is organized as follows: Chapter 2 provides a comprehensive overview of the fundamental topics that our research builds upon. As a baseline, we explain the main principles of the Internet infrastructure and explain the history of digital account and data management. Further, we provide fundamentals of computer security. Subsequent chapters are composed based on a similar structure. Chapter 3 initially systematizes data sovereignty and presents two main contributions: With *Portal*, we secure the processing of ZKPs at public digital infrastructures and propose an SSO alternative with strong data sovereignty guarantees. With our privacy-preserving authentication protocol, called A-PoA, we enhance the privacy of services in sovereign, modern certification ecosystems. Chapter 4 initially systematizes data provenance solutions and presents three main contributions: With *Janus*, we optimize ZKP computations and facilitate the deployment of data provenance protocols in constrained environments. With *Origo*, we optimize the bandwidth requirements of data provenance approaches and facilitate an Internet-wide deployment. With *zkGen*, we automate the generation of PET computation circuits and enable user-driven policies for applications that rely on PETs. Chapter 5 discusses the contributions of this work, answers the research questions, and concludes our work. The appendix presents extended details of ZKP systems and benchmarks, lists the security proofs of our contributions, and states how our publications apply to this thesis.

2. Preliminaries

In this work, we propose new approaches to manage digital identifiers and data privacy, and we optimize protocols that securely verify the provenance of online data. In order to understand different concepts and building blocks of the constructions, we cover a range of computer technologies among different domains. First, we provide the preliminaries to understand how online services operate in today's Internet infrastructure and explain the evolution as well as trends of digital identity management. In addition, we cover emerging decentralized public infrastructures. Afterward, we go through provable security, secure algorithms, and secure protocols to explain the computer security used in this work.

2.1. Internet Infrastructure

The Internet infrastructure is a global body of computers, networks, services, protocols, and applications. We limit our focus to a core subset of services that establish the functionality that lets end devices connect to web services. Subsequently, we explain a selection of protocols that enable the identification and authorization of end devices at web services. Last, we present new computation and data storage paradigms which are possible in emerging public digital infrastructures.

2.1.1. Traditional Internet Services

The Internet is an interconnected set of networks that enables the routing of digital messages between connected computers. To securely communicate with another computer over the Internet, computers invoke several services for the establishment of a secure communication channel. In the following, we introduce protocols and services that are necessary for the communication between end devices and web services.

Internet Protocol

The Internet Protocol (IP) protocol handles the unique identification of devices connected to the Internet. Every device in a network has a unique IP address and every network manages a different range of IP addresses. To share a message, the sending device must know the destination IP address of the other device. If the destination IP address is not part of the local network that manages the IP of the sending device, then routing services forward the message until the network controlling the destination IP is reached. Because IP addresses exist in numerical representations that are difficult to remember, computer addresses rely on easy-to-remember domain names, which can be translated to IP addresses.

Domain Name System

The main purpose of the Domain Name System (DNS) is the resolution and lookup of IP addresses for requested domain names. DNS is a decentralized network of name servers that provide the DNS directory service. With that, name servers maintain a set of resource records for every domain name. On the client side, DNS relies on resolver services that iteratively query name servers with a domain name. Name servers look up a domain name by querying local databases and return associated resource records for the queried domain. If a name service cannot answer a query, then the name server returns a set of other name servers that are likely to answer the query. The DNS resolver at the client continues to query name servers until a domain name is resolved and reports an error if no resources can be found.

If users buy a domain name at a domain name registrar (e.g. Godaddy) and become a domain owner, then users have the ability to manage resource records at self-determined DNS name servers. A possible configuration of a resource record maps an IP address to the domain name. If a mapping between a domain name and an IP address exists, then devices can resolve the destination addresses of other computers by querying DNS. For example, an owner of the domain `example.com` operates a web service at the IP address `168.119.165.10` and registers the tuple (domain, IP) at a DNS name server. In order to talk to the web service with the computer address `example.com`, the device resolves the IP address for the domain via DNS and sends a message with the destination IP address `168.119.165.10` to the Internet. Routing services of the Internet ensure that the message reaches the sub-network, which connects the computer running the web service to the Internet. Due to the fact that online messages encapsulate destination as well as sender addresses, the web service is able to respond back to the device by using the sender address. This way, a two-way communication channel can be established.

Public Key Infrastructure

Reaching another computer over the Internet with a message is a first step. However, to set up a secure communication channel between a device and a web service, additional steps are required. In this paragraph, we explain the Public Key Infrastructure (PKI) of the Internet that authenticates and certifies trustworthy online services. The certification process gives web services access to a cryptographic key pair that can be used to secure and authenticate messages (cf. Public Key Cryptography (PKC) in Section 2.2.2). Without the PKI, end devices cannot reliably check if the communication endpoint is who it claims to be. Neither can end devices establish secure communication channels using PKC.

To obtain a certificate and, with that, a PKC key pair for a web service, domain owners are challenged to prove the ownership of the domain to a PKI registration authority. In this challenge, the registration authority verifies if the operated web service has a valid DNS entry that points to the IP of the web service. The challenge ensures that the domain owner successfully registers the DNS entry that connects the IP address of the web service to the owned domain. If the challenge succeeds, the PKI issues a certificate to the web service via a certificate authority. The PKI certificate contains a digital signature of the certificate authority over the domain and the key pair of the web service. If web services exchange the PKI certificate, end devices can verify if the communication endpoint has been accepted by the trusted PKI service. Web services exchange PKI certificates in a standardized protocol for secure communication (cf. Transport Layer Security (TLS) in Section 2.2.3).

Web Services

Communication parties rely on the TLS protocol to set up a secure communication channel between two parties: The end device and the web service. TLS enables message confidentiality and integrity, and throughout the protocol, devices can verify the authenticity of the web service endpoint. If an external party intercepts the TLS traffic, the external party cannot tamper with the messages. Neither can the external party access the contents of the exchanged messages. We provide further details on TLS in Section 2.2.3.

Once the secure communication channel has been established, devices can securely share or request application-specific data from the web service. If the application at the web service is stateless, then any device is able to access the application at the web service. However, web services typically rely on a state of authentication to identify connected devices. Without any authentication or identifying status, malicious devices would have an easy opportunity to tamper with the application at the web service. As a result, web services require protocols and applications that target the secure authentication and management of online identifiers. In the following, we present a short history of traditional solutions for digital identity management.

2.1.2. Digital Identity Management

In recent years, different forms of Identity Management (IdM) have emerged (cf. Figure 2.1). We elaborate on the benefits and drawbacks of distinct approaches in the following paragraphs.

Centralized Identity

Initially, web services on the Internet individually implemented the registration, authentication, and maintenance of online accounts. However, as the Internet evolved and many new devices and services connected to the Internet, login management for many different services became a burden at end devices. The main technique used to identify in the past and even today remains password and credential-based authentication [20]. Combined with the amount of digital services, password-based authentication led to an increase in credential reuse. Account breaches confirmed the vulnerability of reused, weak passwords. At the same time, the demand grew for a simple custodial solution for identity management.

Federated Identity

Federated identity management solved the usability issue of centralized identity which required remembering, storing, and maintaining authentication credentials at end devices. Federated identity introduced a trusted custodial party that takes over the management of

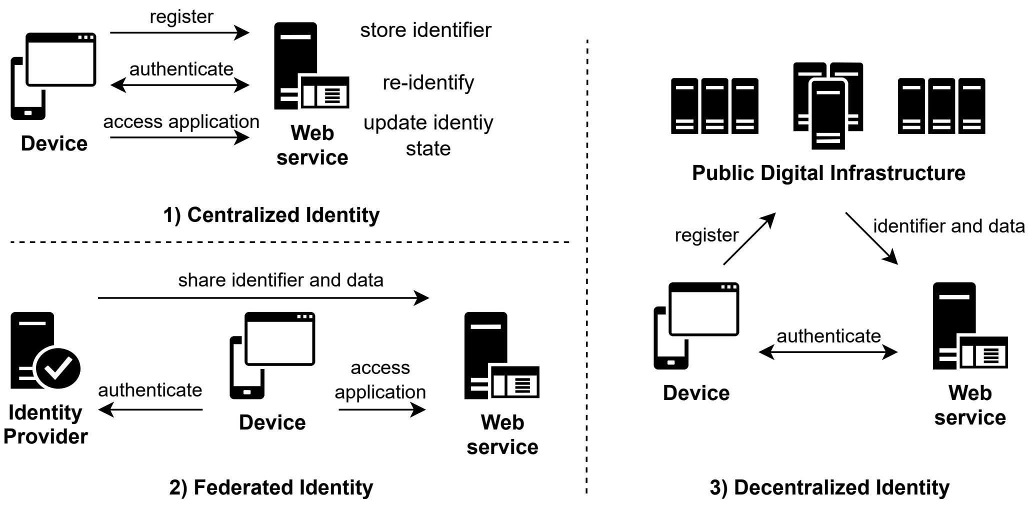


Figure 2.1.: Overview of digital identity approaches, where digital identifiers and associated data is managed either (i) at the web service itself, (ii) at dedicated identity services, or (iii) at public digital infrastructures.

identifiers and data attributes for end devices. Additionally, the trusted custodial party takes over the authentication and the generation of identifying data on behalf of web services (cf. graph 2 of Figure 2.1). With that, federated identity brought the benefits of exchanging and linking identifiers, established itself, and, until today, counts as the go-to standard to authenticate online. The most popular protocol that reflects the paradigm of federated identity is the concept of SSO with Open Authorization 2.0 (OAuth 2.0), where users delegate the authority of administering identifiers and data access to an external and trusted Identity Provider (IdP) [21].

The disadvantages of federated identity emerged quickly. With few centralized identity providers managing millions of online accounts, federated identity generated attractive data hubs for attackers [22]. The resulting data breaches confirmed emerging data privacy concerns of end device tracking and profiling for commercial purposes [23], [24]. The transparency issues caused by federated identity eventually violated fundamental human rights [2], which accelerated legislative regulations of online privacy through governments. As a result, computer research investigates novel approaches to identity management to alleviate custodial data controllers from liability and compliance penalties. One of the most researched domains of future online identity considers a decentralized setting, where data sovereignty is brought back to end devices.

Decentralized Identity

Instead of relying on centralized IDPs, the setting of decentralized identity outsources the administration of identifiers and associated data to decentralized public infrastructures. Decentralized public infrastructures occur as blockchains (e.g. Ethereum [25]), where code deployments in the form of smart contracts implement the functionalities of identity management. As such, smart contracts already implement data resolution architectures through services such as the Ethereum Name System (ENS) [26]. Thus, when devices authenticate towards web services with a decentralized sign-on alternative called Sign-In with Ethereum (SIWE) [27], then web services obtain an identifier that links to ENS data of devices (cf. graph 3 of Figure 2.1).

The benefit of the decentralized approach is that operators of the blockchain network cannot easily track or profile device behavior. Because, web services can freely decide which blockchain operator to contact in order to request smart contract data. Beyond better privacy opportunities, decentralized sign-on gives devices access to a cryptographic PKC key pair. The key pair is used to control updates at data administration contracts and gives devices the sovereignty to determine which data to upload and share. One important drawback of today's public digital infrastructures is that access as well as compatibility with privacy-empowering technologies remains a research area and does not exist in production yet. As a result, this thesis proposes new approaches to secure data privacy in the context of public digital infrastructures for identity management. We discuss further details of blockchains in the upcoming Section 2.1.3.

Credential Management

Digital identifiers allow the discovery, association, and linking of online accounts and data. But, out of the box, digital identifiers do not provide trust. In order to build up trust, online identities are required to collect evidence, recommendations, or reputations, which, by the nature of online interactions, are difficult to establish. Digital interactions mostly rely on two-party sessions, where witnessing online behavior is difficult or even impossible to perform by other parties or services. Most frequently, devices build up reputation-based trust in individual online services. In this scenario, it remains the choice of the web service to recommend an account or share evidence with other online participants or to the public. However, if web services are willing to contribute to the establishment of a trusted identity, then web services issue credentials towards the subject of trust.

Credential management covers (i) the issuance of credentials between the credential issuer and the credential holder, and (ii) the delegation of credentials between different holders. Digital credentials are signed data relations between two digital identities that an external third party can verify. Usually, issuers confirm the information the credential is attesting to before granting credentials. Depending on the data certification by a credential, the influence and trust of online identities grows. Due to the fact that credentials require digital signatures, credential ecosystems depend on key management infrastructures (e.g. the PKI). In the decentralized identity setting, digital credentials have been reconsidered, where the World Wide Web Consortium (W3C) standardized the notion of Verifiable Credentials (VCs) [28] and Decentralized Identifiers (DIDs) [29]. Today, approaches exist that connect VCs to SSO protocols such as *OpenID Connect* [30].

The VC ecosystem utilizes DIDs to connect credentials to a holder (cf. Figure 2.2). DIDs are universally unique identifiers with a mandatory prefix, method, and method-specific identifier, each separated by colons (e.g. *did:web:example.com*). Each DID resolves to a DID document

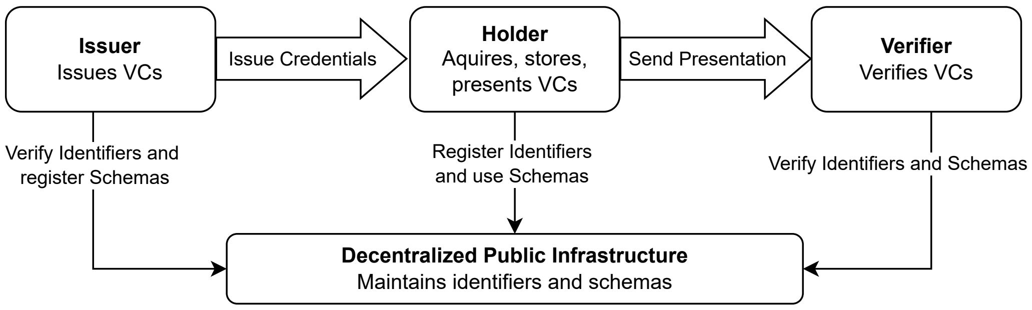

Issuer
Issues VCs
Issue Credentials
Holder
Aquires, stores, presents VCs
Send Presentation
Verifier
Verifies VCs
Verify Identifiers and register Schemas
Register Identifiers and use Schemas
Verify Identifiers and Schemas
Decentralized Public Infrastructure
Maintains identifiers and schemas

Figure 2.2.: Overview of the Verifiable Credentials (VCs) ecosystem. This figure has been recreated based on the illustration provided by the VC standard [31].

that specifies security-specific parameters and other mechanics (associated VCs, claims on the subject, etc.). Issuing parties with the ability to write to a decentralized public infrastructure must register their DID. VC holders do not register their DID at the decentralized public infrastructure. However, VC holders disclose their did during credential presentations to a third-party verifier. To protect their privacy, credential holders can switch between different DIDs to obfuscate tracking of their own identity. To issue a VC, registered authorities (e.g. Root Authority (RA) and Credential Issuing Authority (CIA)) need to register a Credential Schema (CS) and Credential Definition (CD) at the verifiable registry. Upon reception of a VC, which links to a CD and CS, a VC holder is able to prove claims found in the VC in a privacy-preserving manner [32]. VC verifiers leverage the decentralized public infrastructure to resolve and validate VC entries, signatures, and DID identifiers.

2.1.3. Decentralized Public Infrastructures

Open networks in the form of decentralized public infrastructures emerged with the advent of blockchain protocols (e.g. Bitcoin went live in 2009 [33] and Ethereum went live in 2012 [25]). Blockchains are publicly accessible computation networks anyone can join. Further, anyone can transparently verify the computations happening on blockchains due to the cryptography that is baked into blockchains. In addition to blockchains, decentralized storage networks such as the Interplanetary File System (IPFS) [34] brought publicly accessible and verifiable storage. In the following, we provide further details on how cryptography influences novel concepts of decentralized public infrastructures.

Blockchains & Smart Contracts

Public blockchains are open computer networks anyone can join, which run a *consensus* protocol to agree upon a common and correct *state* s_t at time t. The *state* maintains two types of accounts. The Externally Owned Account (EOA) is controlled by a PKC key pair and is updated if a *user* owning the key pair sends signed transactions to the blockchain. The second type of an account is called *smart contract*, which is an executable program at a unique address that can be invoked by transactions. Transactions are issued by *users* and are stored in a memory pool called *mempool* until the *consensus* protocol proposes a new state update via a new block of transactions. If a new state update is proposed via a new block of transactions, then blockchain nodes apply transactions, process *smart contracts* in virtual machines, and compare the local *state* updates with the digests provided in the new block. If the verification succeeds, blockchain nodes locally apply the *state* update and, with that, reach a new global state agreement.

The execution of *smart contracts* is measured in *gas* and must be paid by a medium called cryptocurrency. Cryptocurrencies are collected by operators of the blockchain network and incentivize the correct maintenance of the *state* through the *consensus* construction that ensures the following principles. *State* updates and even a new chain of blocks are accepted if the new block complies with the longest available chain of blocks. Additionally, the security of the *state* depends on a cryptographic challenge that operators must solve to earn cryptocurrencies. The cryptographic challenge prevents an attacker from proposing a forged chain to the network of operators. Because, blockchains rely on the assumption that the attacker cannot spend or access more resources for solving the cryptographic challenge than the network of operators over a longer period of time.

As such, blockchains obtain *state* integrity and achieve the properties of *safety*, *liveness*, *consistency*, and, due to the *state* redundancy at every node, *fault-tolerance*. *Safety* provides *state* integrity according to past *states* and prevents adversaries from injecting invalid transactions. *Liveness* ensures that every transaction is eventually included in the *state*. *Consistency* guarantees that every node eventually has the same view of the *state*. It holds that every recorded transaction becomes immutable and persists within the *state* and that transactions achieve *non-repudiation*, where the signature of every transaction unambiguously identifies a *user*.

Decentralized Storage

Decentralized storage exists via the prominent example of IPFS as a Peer-to-peer (P2P) network which handles the distributed storage and sharing of data [34]. Data is made addressable via cryptography, where hash functions generate hash-based content addressing in a global namespace (cf. Secure Hash Functions in Section 2.2.2). The global namespace is hosted by a group of operators that anyone can join. IPFS provides persistent versioning and gives devices the ability to track past changes. Due to the properties of content addressable storage and persistent file versioning, IPFS extends key features of blockchain networks and integrates itself seamlessly into decentralized public ecosystems.

2.2. Security Fundamentals

This thesis deals with the advancement of privacy-enhancing technologies (PETs) in the context of Internet security protocols. We introduce a subset of PETs that are relevant to understand the contributions of this work. We structure our security fundamentals into the topics of provable security, security algorithms, and security protocols. The topic of provable security outlines how computer security can be modeled and defines core principles, assumptions, and components. With the topic of security algorithms, we show how algorithms express security properties on parameters. Last, we explain how the security protocols used in this thesis can be composed based on the previously defined security algorithms.

2.2.1. Provable Security

This work uses the concept of provable security to show the soundness of newly introduced security contributions for PETs. Provable security can be constructed with different techniques and in different security models. However, in any case, provable security depends on an adversarial model with access to a system model. In the adversarial model, the attacker is given access to computational resources and the system model, which formally defines the security requirements of a computing task. The security requirements, which the attacker must break, depend on computational hardness assumptions. A mathematical and formal proof of security is obtained, if the evaluation of a computation task performed by the adversary can be reduced to computing the hardness assumptions. In the following, we define the necessary models to explain the techniques of provable security.

Security Models

Security models define a setting with different assumptions in which the algorithm can be proven secure. In the strongest security model, an algorithm is only secure if no additional assumptions are required. This implies that the adversary has access to as much information as possible and is computationally unbounded, and algorithms achieve information-theoretic security (perfect security). The standard security model assumes that adversaries are computationally bounded, which implies that the algorithm achieves computational security. The random oracle security model assumes that the adversary has access to a black box. It holds that a query to the black box with new inputs yields unpredictable outputs. Notice that output collisions are possible but the unpredictability of outputs always holds. Further, security models such as the common reference string (CRS) model or the PKI model assume that all parties in the system have access to specific security parameters. In the CRS model, everyone has access to a random string from a pre-determined distribution. In the PKI model, all parties possess PKI certificates and PKC key pairs such that messages can be authenticated.

System Model

The purpose of the system model is to define security-related assumptions that hold when parties execute a computation task. The description of the task itself is not part of the system model. Instead, the security proof technique defines the computation task and functionality in a specific representation to simplify the understanding. Hence, the system model states the number and roles of parties in the system. In addition, the system model defines security properties as system goals.

Concerning the communication model, parties can communicate fully synchronously with no message delay and with a globally synchronized clock. Or, parties communicate partially synchronously, where the message transmission takes a fixed amount of time and delivery is guaranteed. For asynchronous communication, it holds that messages can be arbitrarily delayed. However, in this case, messages are assumed to be delivered eventually. The system model further defines if messages are authenticated, keep data confidential, or preserve integrity during the transmission. For example, if a system model uses mutual TLS for the communication, then message integrity, confidentiality, and authenticity hold.

Last, the system model selects a security model in which the computation task is proven to be secure.

Adversarial Model

The adversary model defines multiple capabilities of the adversary. Concerning computational power, an adversary is either computationally bounded and capable of computing in polynomial time, or an adversary can compute with unbounded computational resources. The capability to compute must match the security model used by the system. For example, in the strongest security model, an adversary has unbounded compute resources and in the standard model, the adversary has computationally bounded resources.

Concerning the corruption of parties in the system, an attacker can be categorized as fail-stop, passive, active, or semi-malicious. If the adversary corrupts a party to fail-stop, then the adversary is causing a random crash. A passive adversary is an eavesdropper, who is able to observe parties and intercept the communication transcript in order to learn some secret information. Active adversaries corrupt parties such that the parties arbitrarily deviate from the protocol specification. Semi-malicious adversaries honestly follow the protocol specification but select inputs or make decisions arbitrarily. We do not cover additional corruption models such as covert or rushing adversaries but mention them here for completeness.

The adversary access to the system model is either static, where the number of corrupted parties is determined initially and never changed thereafter. If the attacker has adaptive access to the system, then the adversary can choose the number of compromised parties at any time during the protocol execution.

Algorithm Security

We cover multiple aspects of secure algorithms in the following paragraphs.

Notions of Security With regard to the computational power of the adversary, a secure algorithm can be classified with respect to providing information-theoretic security, statistical security, and computational security. The information-theoretic model assumes computationally unbounded adversaries whereas computational security polynomial time adversaries assumes. For statistical security, the adversary has an advantage of less or equal to $\frac{1}{2} + \epsilon$ probability in correctly solving the computation task. For computational security, the algorithm must be resistant against computationally bounded adversaries (e.g. polynomial time adversaries). In other words, computational security is resistant to brute-force attacks. Further, semantic security (computational or statistical indistinguishability) prevents an adversary from approximately differentiating a message distribution with a probabilistic guess. Again, the selection of the security model determines the type of security an algorithm must achieve.

Hardness Assumptions: Security algorithms depend on mathematical problems (hardness assumptions), which are difficult to compute. For example, computing the factorization of integers or the discrete log on specific mathematical parameters is assumed to be infeasible if the adversary has bounded computation resources. With access to quantum computing, the adversary can break the factorization of integers but cannot break the hardness of lattice-based computing tasks. Hardness assumptions can be defined in different mathematical domains. For example, group theory is used to express the discrete log problem and number theory expresses the Rivest–Shamir–Adleman (RSA) hardness assumptions. Linear algebra is used to define lattice-based problems. Algebraic geometry and group theory are used to define the underlying hardness assumptions of Clliptic Curve (EC) security. To clarify, we list different hardness assumptions next which are assumed to be infeasible to break. Further assumptions and details can be found in the work [35].

- (Discrete Log) **Discrete Logarithm Problem (DLP):** Given the group elements $g, h \in \mathbb{G}$ with $h = g^x \mod N$, is is difficult to find $x = \log_g h$. $\mathbb{G}$ is a finite cyclic group.
- (Discrete Log) **Elliptic Curve Discrete Log Problem (ECDLP):** Given the points of an elliptic curve E over a finite field $\mathbb{F}_q$ with $q = p^n$ and prime p as $P, Q \in E(\mathbb{F}_q)$, it is infeasible to find an integer a that determines $Q = aP$.
- (Discrete Log) **Computational Diffie-Hellman (CDH):** Given the group elements $g, g^a, g^b \in \mathbb{G}$, no polynomial time adversary can compute g^{ab}.
- (Discrete Log) **Decision Diffie-Hellman (DDH):** Given $h, g^a, g^b \in \mathbb{G}$, it is computational infeasible (polynomial time adversary) to determine if $h \stackrel{?}{=} g^{ab}$.
- (Factoring) **RSA Problem (RSAP):** Given two large primes p, q (with bits >= 200) and $n = pq$, it is infeasible to factorize n for p, q. Finding p, q is possible with primality testing algorithms.

Cryptosystems: Based on the type of hardness assumption, different cryptosystems are available today. Even though cryptosystems do not necessarily rely on the same hardness problem, cryptosystems can define equal security functionalities through security algorithms (cf. Section 2.2.2). For example, a digital signature scheme can be instantiated using an ElGamal or EC (discrete log), a RSA (factoring), or an Learning with Errors (LWE) (lattice) based cryptosystem.

Cryptographic Primitives: Secure algorithms are classified based on the function design or primitive, which decides when and how a hardness assumption is used to protect variables. For example, one-way functions f are designed to protect inputs during the function evaluation. It holds that the evaluation of f is easy to compute on the inputs x. However, with access to the output $f(x)$, one cannot compute the inverse of f to determine the input x. Trapdoor functions are one-way functions with a special secret that allows the computation of the inverse. It holds that every trapdoor function is a one-way function but not every one-way function is a trapdoor function. For example, trapdoor functions exist in asymmetric cryptography (cf. PKC in Section 2.2.2), where, by knowing a public key, computing a ciphertext is easy. But to decrypt a ciphertext, a party must have access to the corresponding private key. Similar to hash functions, symmetric cryptography uses one-way functions that do not count as trapdoor functions. Typical hash functions are one-way without a trapdoor because hash functions cannot be reverted at all. Concerning symmetric cryptography, computing a ciphertext using Advanced Encryption Standard (AES) requires knowledge of a secret key but, with the key, it is equally easy to compute the inverse of AES. Hence, AES is a one-way function but does not count as a trapdoor function.

Pseudo-random Functions (PRFs) are keyed or seeded functions $\mathcal{F}_k$: $k \times \{0,1\}^* \rightarrow \{0,1\}^*$, where the key $k = \{0,1\}^*$ is uniform and $\mathcal{F}_k$ is indistinguishable from a function chosen uniformly at random from a set of functions with equal distributions (same input-output domains). Pseudo-random Generators (PRGs) are PRFs but expect a shorter uniform seed or key, which is then turned into a longer pseudo-random output. PRGs continuously add entropy to an internal pseudo-random state, where entropy is typically captured from physical hardware states. PRGs are used in key derivation functions for encryption algorithms.

Last, we introduce the principle of permutation functions. A permutation p on a set X is a bijective function from X to p. A trapdoor one-way permutation additionally implements the model of a trapdoor one-way function and can be used to model asymmetric encryption schemes. Pseudo-random Permutations (PRPs) can be used to model symmetric encryption algorithms such as AES. We introduce additional cryptographic principles throughout the Section 2.2.2, where we cover the techniques of MPC, ZKP, hash functions, commitments, oblivious transfer, etc.

Security Proof Techniques

All security-proof techniques rely on an adversary model with access to a system model. Based on the defined capabilities, the adversary obtains a set of parameters through interaction with the system model, which executes a computation challenge in the context of a security model. Next, the security proof technique evaluates if the computation challenge at the adversary can be reduced to computing hardness assumptions. In the following, we introduce different techniques to formally evaluate the security of algorithms or protocols.

- **Game-based Security Proof:** This type of security proof considers a game between the adversary and a made-up challenger. Both parties are assumed to have probabilistic processes that can communicate. The game is modeled as a probability space which defines an advantage scenario for the adversary. The advantage is expressed via a comparison of two probabilities: a target probability and a probability that an event occurs. The security is proven if the probability of the event is negligibly close to the target probability.

The event is caused by a polynomial-time adversary $\mathcal{A}_a$ who solves the challenge of breaking the security of an algorithm a with non-negligible probability. Next, as a target probability, game-based security proofs rely on the challenger $\mathcal{C}_h$ who is capable of breaking a hardness assumption h with non-negligible probability. This is possible if the challenger can run the security algorithm with the secret inputs that define the hardness assumption. Now, if both $\mathcal{A}_a$ and $\mathcal{C}_h$ are used in the security game, then the probability space consists of (i) the event that $\mathcal{A}_a$ succeeds (security property is broken) and (ii) the event that $\mathcal{C}_h$ succeeds (hardness assumption is broken). The subtraction of both probabilities defines the advantage of the adversary. A negligible advantage means that breaking the security property in the game is as likely as solving a hardness assumption with a non-negligible probability. Since breaking a hardness assumption is assumed to be infeasible, it holds that the security algorithm is secure. A popular model to construct game-based security proofs is the game-playing framework [36], [37].

- **Simulation-based Security Proof:** In the simulation-based paradigm, the adversary is given access to a (i) real protocol execution T_{real} and a (ii) ideal protocol execution T_{ideal}. In the real world, the adversary captures a protocol execution T_{real} with the capabilities defined in the adversarial model. Here, no additional assumptions are required. The ideal world relies on an incorruptible trusted party that executes the ideal functionality of the protocol. To run the ideal protocol execution, an adversary called a simulator generates and collects all inputs of honest and dishonest parties and runs the protocol over all collected inputs. Then, the simulator adds all messages of the protocol to T_{ideal}. The purpose of running the protocol at the simulator is the collection of intermediate messages. Afterward, the simulator invokes the ideal functionality with all required inputs and, subsequently, receives and adds the outputs to T_{ideal}. If the adversary cannot distinguish the protocol transcript of the real world T_{real} from the protocol transcript that is generated in the ideal world T_{ideal}, then the adversary learns nothing new (semantic security) and security holds.

- **Universal Composability Security Proof:** The above-mentioned security proof techniques cover stand-alone executions of security protocols or algorithms. However, if an algorithm is composed of multiple secure algorithms (which might run concurrently), then a stand-alone security proof is not enough. The Universal Composability (UC) model can be used to show the security of protocols that rely on a composition of secure algorithms. The UC model adds another adversarial entity, called the environment, to the simulation-based model. The environment captures the inputs and outputs of all parties and interacts with the adversary. The security in the UC model holds if no environment can distinguish a real protocol execution with the adversary from an ideal protocol execution with a simulator.

2.2.2. Secure Algorithms

In this section, we describe security algorithms and functionalities at the abstraction level of algorithmic building blocks. Notice that the building blocks can be instantiated with different cryptosystems (cf. Section 2.2). Depending on the used cryptosystem, the parameterization of the security algorithm can deviate. For each security algorithm and principle, we present the security properties and guarantees.

Secure Hash Functions

A secure hash function implements an algorithm, where

- **h.Hash**$(m) \rightarrow (h)$ takes as input a message string and outputs a constant size string h.

Hash functions guarantee three properties: *Preimage-resistance* ensures that given h, and attacker cannot find m if $h = \textbf{h.Hash}(m)$. *Second preimage-resistance* ensures that given m_1 an attacker cannot find m_2 such that **h.Hash**(m_1)=**h.Hash**(m_2) holds, with $m_1 \neq m_2$. *Collision-resistance* ensures that finding $m_1 \neq m_2$ with **h.Hash**(m_1)=**h.Hash**(m_2) is infeasible.

Keyed-hash or Hash-based Key Derivation Function

A HMAC-based Key Derivation Function (HKDF) function converts parameters with insufficient randomness into suitable keying material for encryption or authentication algorithms. The HKDF scheme is defined by a tuple of algorithms, where

- **hkdf.ext**$(s_{\text{salt}}, k_{\text{ikm}}) \rightarrow (k_{\text{pr}})$ takes in a string s_{salt}, input key material k_{ikm}, and returns a pseudorandom key k_{pr}.
- **hkdf.exp**$(k_{\text{pr}}, s_{\text{info}}, l) \rightarrow (k_{\text{okm}})$ takes in a pseudorandom key k_{pr}, a string s_{info} and a length parameter l and returns output key material k_{okm} of length l.

Both functions **hkdf.ext** and **hkdf.exp** internally use the **hmac** algorithm (cf. Formula 2.1), which takes in a key k, a bit string m, and generates a string which is indistinguishable from uniform random strings. The **hmac** algorithm requires a hash function H with input size b (e.g. b=64 if H=SHA256).

$$\begin{aligned}\textbf{hmac}(k, m) =& H((k' \oplus opad)||H((k' \oplus ipad)||m))\\ & \text{with } k' = H(k), \text{ if } len(k) > b \\ & \text{and } k' = k, \text{ else}\end{aligned} \tag{2.1}$$

Authenticated Encryption

Authenticated Encryption with Associated Data (AEAD) provides communication channels with *confidentiality* and *integrity*. This means that exchanged communication records can only be read by parties with the encryption key, and modifications of encrypted data can be detected. An AEAD encryption scheme is defined by the following tuple of algorithms, where

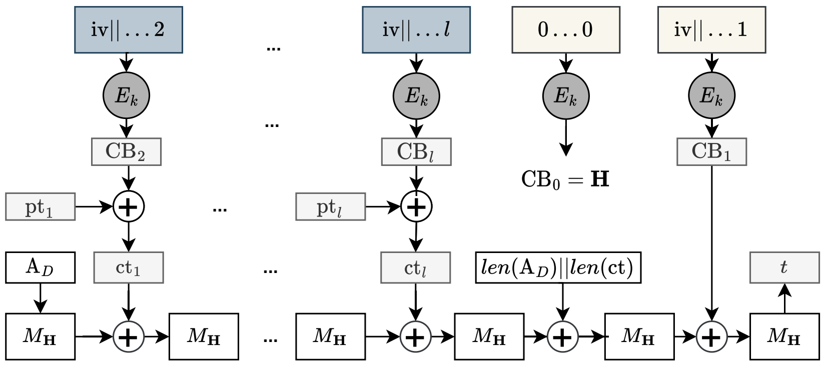


Figure 2.3.: AEAD stream cipher configured with AES in the Galois/Counter mode (Galois Counter Mode (GCM)). The algorithm encrypts a plaintext $\mathbf{pt} = [pt_1, \ldots, pt_l]$ to a ciphertext $\mathbf{ct} = [ct_1, \ldots, ct_l]$ under key k and authenticates the ciphertext **ct** and associated data A_D with the tag t. The symbol $M_\mathbf{H}$ is a Galois field multiplication which translates bit strings into GF(2^{128}) polynomials, multiplies the polynomials modulo the field size, and translates the polynomial back to the bit string representation.

- **aead.Setup**$(1^\lambda) \rightarrow (\mathrm{pp_{aead}})$ takes in the security parameter λ and outputs public parameters $\mathrm{pp_{aead}}$ of a stream cipher scheme E and authentication scheme A.
- **aead.Seal**$(\mathrm{pp_{aead}},\ pt,\ k,\ a_D) \rightarrow (ct,\ t)$ takes in $\mathrm{pp_{aead}}$, a plaintext pt, a key k, and additional data a_D. The output is a ciphertext-tag pair (ct, t), where $ct = E(pt)$ and $t = A(pt, k, a_D, ct)$ authenticates ct.
- **aead.Open**$(\mathrm{pp_{aead}},\ ct,\ t,\ k,\ a_D) \rightarrow \{pt,\ \varnothing\}$ takes in $\mathrm{pp_{aead}}$, a ciphertext ct, a tag t, a key k, and additional data a_D. The algorithm returns the plaintext pt upon successful decryption and validation of the ciphertext-tag pair, otherwise it returns an empty set $\varnothing$.

Stream ciphers are characterized by pseudorandom generators (AES in the GCM mode), which incrementally output key streams or Counter Blocks (CBs) (cf. Figure 2.3). CBs are combined with plaintext data chunks to compute ciphertext data chunks. Subsequently, AEAD ciphers compute an authenticated tag t on all ciphertext chunks and associated data.

Cryptographic Accumulator

One-way cryptographic accumulators, as initially introduced in [38], provide the ability to verify the set membership of an element $x_i \in X$, with $i = \{1, 2, \ldots, N\}$, where N is the number of elements within a set X, without revealing individual elements of the set. Throughout this work, we require the accumulator to be dynamic and positive. Dynamic accumulators support additive and subtractive operations which increase and decrease the number of elements of the accumulator respectively [39]. Positive accumulators support proof of membership.

Among multiple options of accumulator types, this work relies on the RSA-accumulator of the work in [40] due to the following reasons:

- An accumulator based on modular exponentiation has minimal storage requirements and *O(1)* verification complexity compared to Merkle-tree accumulators [41], [42].
- The RSA-accumulator can be used in combination with the group $\mathbb{Z}_n^*/\{\pm 1\}$, with the RSA modulus n. More specifically, it relies on the strong RSA assumption [43], and the hardness of the discrete log problem [44] which makes it applicable to succinct Proof of Knowledge (PoK) of a discrete-log schemes, as introduced in the work [45].

The work in [40] defines the accumulator value as a quadratic residue a_t modulo n at time t, with $n = p * q$ as the RSA modulus. The value a is initialized through the generator $g_{\text{acc}} \in \mathbb{QR}_n$, where $\mathbb{QR}_n$ is the subgroup of quadratic residues of the generic group of unknown order $\mathbb{G}_?$. The security of the RSA-accumulator follows the strong RSA assumption with primes p, q, p', and q', with $p = 2p' + 1$ and $q = 2q' + 1$. Accumulator elements are odd positive prime integers because otherwise, an element could be proven a member of the set of elements even though it is not (exclusion requirement of element divisors). Adding an element $x_i \in X_t$, with $X_t = \{x_1, x_2, ..., x_i\}$, to the accumulator works by calculating $a_{t+1} = a_t^{x_i} \mod n$. The extraction of the respective witness w_t for x_i calculates as $w_t = a_t^{X_t \setminus \{x_i\}} \mod n$. A successful verification of the element x_i with its witness w_t as $a = w_t^{x_i} \mod n$ equals the latest accumulator value a_{t+1}. Deleting an element x_i from the set of elements in the accumulator requires the knowledge of the factorization of n. The deletion works by calculating $a_{t+1} = a_t^{x_i^{-1} \mod \phi(n)} \mod n$ and updating another witness w_t paired with x calculates as $w_{t+1} = w_t^c a_{t+1}^b \mod n$ and relies on the Bezout coefficients b and c, with $bx + cx_i = 1$.

Multiplicative to Additive (MtA) Conversion based on Homomorphic Encryption

The secure 2PC MtA algorithms convert multiplicative shares x, y into additive shares α, β such that $\alpha + \beta = x \cdot y = r$ yield the same result r. The MtA algorithms exist in a vector form, which maps two vectors $\mathbf{x}, \mathbf{y}$, with a product $r = \mathbf{x} \cdot \mathbf{y}$, to two scalar values α, β, where the sum $r = \alpha + \beta$ is equal to the product r. The functionality of the vector MtA scheme can be instantiated based on Paillier additive Homomorphic Encryption (HE) [46]. Additive HE allows parties to locally compute additions and scalar multiplications on encrypted values. With the functionality provided by the Paillier cryptosystem, we define the vector MtA scheme, as specified in the work [47], with the following tuple of algorithms, where

- **mta.Setup**(1^λ) $\rightarrow$ (sk_P,pk_P) takes in the security parameter λ and outputs a Paillier key pair (sk_P,pk_P).
- **mta.Enc**($\mathbf{x}$,sk_P) $\rightarrow$ (**c1**) takes in a vector of field elements $\mathbf{x}$=$[x_1, \ldots, x_l]$ and a private key sk_P and outputs a vector of ciphertexts **c1**=$[E_{sk_P}(x_1), \ldots, E_{sk_P}(x_l)]$.

- **mta.Eval**(**c1**,**y**,pk_P) $\rightarrow$ (c_2,β) takes in the vector of ciphertexts **c1**=$[\mathrm{c1}_1, \ldots, \mathrm{c1}_l]$, a vector of field elements **y**=$[y_1, \ldots, y_l]$, and a public key pk_P. The output is a tuple of a ciphertext $\mathrm{c2} = \mathrm{c1}_1^{y_1} \cdot \ldots \cdot \mathrm{c1}_l^{y_l} \cdot E_{pk_P}(\beta')$ and the share $\beta = -\beta'$, where $\beta' \xleftarrow{\$} \mathbb{Z}_p$.
- **mta.Dec**(c2,sk_P) $\rightarrow$ (α) takes as input a ciphertext c2 and a private key sk_P and outputs the share α=$D_{sk_P}(\mathrm{c2})$.

The tuple of algorithms is supposed to be executed in the order where party p_1 first calls **mta.Setup** and **mta.Enc**. The function $E_k(z)$ is a Paillier encryption of message z under key k. After p_1 shares the public key pk_P and the vector of ciphertexts **c1** with party p_2, then p_2 calls **mta.Eval** and shares the ciphertext c2 with p_1. Last, p_1 calls **mta.Dec**, where $D_k(z)$ is a Paillier decryption of message z under key k. If the algorithms are executed in the described order, then party p_1 inputs private multiplicative shares in the vector **x** and obtains the additive share α. Party p_2 inputs the private vector of multiplicative shares **y** and obtains the additive share β. In the end, the relation $\mathbf{x} \cdot \mathbf{y} = \alpha + \beta$ holds, and neither the party p_1 nor the party p_2 learns anything about the private inputs of the counterparty.

Oblivious Transfer (OT)

The 1-out-of-2 OT_2^1 protocol involves two parties where party p_1 sends two messages m_1, m_2 to party p_2 and p_1 does not learn which of the two messages m_b is revealed to party p_2. Party p_2 inputs a secret bit b that decides the selection of the message m_b. An Oblivious Transfer (OT) scheme is defined by a tuple of algorithms, where

- **ot.Setup**(1^λ) $\rightarrow$ ($\mathrm{pp}_{\mathrm{OT}}$) takes as input a security parameter λ and outputs public parameters $\mathrm{pp}_{\mathrm{OT}}$ of a hash function H and encryption schemes, where E_1/D_1 encrypts/decrypts based on modular exponentiation and E_2/D_2 encrypts/decrypts with a block cipher.
- **ot.TransferX**($\mathrm{pp}_{\mathrm{OT}}$) $\rightarrow$ (X) takes in $\mathrm{pp}_{\mathrm{OT}}$, samples $x \xleftarrow{\$} \mathbb{Z}_p$, and outputs an encrypted secret $X = E_1(x)$.
- **ot.TransferY**($\mathrm{pp}_{\mathrm{OT}}$, X, b) $\rightarrow$ (Y, k_D) takes in $\mathrm{pp}_{\mathrm{OT}}$, a cipher X, a bit b, and samples $y \xleftarrow{\$} \mathbb{Z}_p$. The output is a decryption key $k_D = X^y$ and a cipher Y encrypting as $Y = E_1(y)$ if $b \stackrel{?}{=} 0$, or as $Y = X \cdot E_1(y)$ if $b \stackrel{?}{=} 1$.
- **ot.Encrypt**($\mathrm{pp}_{\mathrm{OT}}$, X, Y, m_1, m_2, x) $\rightarrow$ (**Z**) takes in $\mathrm{pp}_{\mathrm{OT}}$, Y, and derives $k_1 = H(Y^x)$, $k_2 = H((\frac{Y}{X})^x)$. The output is a vector of ciphers $\mathbf{Z} = [E_2(m_1,k_1), E_2(m_2,k_2)]$.
- **ot.Decrypt**($\mathrm{pp}_{\mathrm{OT}}$, **Z**, k_D, b) $\rightarrow$ (m_b) takes in $\mathrm{pp}_{\mathrm{OT}}$, key k_D, the bit b, and a vector of ciphers $\mathbf{Z} = [Z_1, Z_2]$. The output is the message $m_b = D_2(Z_b, k_D)$.

In the OT_2^1 protocol, party p_1 calls **ot.Setup** and **ot.TransferX**, and sends the public parameters and cipher X to p_2. Party p_2 calls **ot.TransferY**, locally keeps the decryption key, and shares the cipher Y with p_1. Now, p_1 shares the output of **ot.Encrypt** with p_2, who obtains m_b by calling **ot.Decrypt**.

Secure 2PC **with Garbled Circuits**

Secure 2PC based on boolean Garbled Circuits (GCs) depends on the 1-out-of-2 OT_2^1 sub-protocol to secretly exchange input parameters of the circuit [48]. We define secure 2PC based on boolean garbled circuits by extending our OT definition with the following tuple of algorithms, where

- **gc.Setup**$(1^\lambda) \rightarrow (\mathrm{pp}_{\mathrm{GC}})$ takes in the security parameter λ and outputs public parameters $\mathrm{pp}_{\mathrm{GC}}$.
- **gc.Garble**$(\mathrm{pp}_{\mathrm{GC}}, \mathcal{C}_G, d_{\mathrm{in}}) \rightarrow (\mathbf{k}_{in}^g, \mathbf{e}, \boldsymbol{G}(\mathcal{C}), T_{\text{k-d}}, T_{\text{d-k}})$ takes as input $\mathrm{pp}_{\mathrm{GC}}$, a boolean circuit $\mathcal{C}_G$, the input bit string d_{in}, and randomly samples signal bits and wire keys $\sigma, k \xleftarrow{\$} \mathbb{Z}_n$. Every wire receives two wire keys where the internal labels map wire keys to the numbers 0 and 1. Based on the signal bits and internal labels, every wire receives two external labels. The output consists of input wire keys $\mathbf{k}_{in}$, the garbled tables $\boldsymbol{G}(\mathcal{C})$, input and output decoding tables $T_{\text{d-k}}, T_{\text{k-d}}$, and external labels $\mathbf{e}$.
- **gc.Evaluate**$(\mathrm{pp}_{\mathrm{GC}}, \mathbf{k}_{in}^g, \mathbf{k}_{in}^e, \mathbf{e}, \boldsymbol{G}(\mathcal{C})) \rightarrow (k_{out})$ takes in public parameters, input wire keys, external labels, and the garbled circuit tables and outputs output wire keys.

On a high level, a 2PC system based on boolean garbled circuits involves a party p_1 as the garbler and party p_2 as the evaluator. Party p_1 calls **gc.Setup** and **gc.Garble**. Subsequently, p_1 sends $\mathbf{e}$, $\mathbf{k}_{in}^g$, $\boldsymbol{G}(\mathcal{C})$, and $T_{\text{k-d}}$ to p_2. If the semi-honest 2PC system is used in the context of an Honest Verifier Zero-knowledge (HVZK) proof system, then p_1 does not share $T_{\text{k-d}}$. Next, to obtain the remaining input labels $\mathbf{k}_{in}^e$ of the evaluator p_2, p_1 and p_2 interact with the OT_2^1 scheme. Initially, both parties call the transfer functions. Next, p_1 sends input wire keys encrypted by **ot.Encrypt** as messages (m_1=$\hat{k}_{\mathrm{in}}^e$, m_2=$\hat{k}_{\mathrm{in}}^{\neg e}$) to p_2. Party p_2 obtains labels k_{in}^e by calling **ot.Decrypt**. Then, p_2 calls **gc.Evaluate** and if $T_{\text{k-d}}$ has been shared, decodes output wire keys to obtain the output data bit string d_{out}.

Commitment Schemes

We formally define cryptographic commitments with the following tuple of algorithms, where

- **c.Commit**$(m, r_c) \rightarrow (c)$ takes in a string m and commit randomness $r_c \xleftarrow{\$} \mathcal{R}$ and yields a commitment string c.
- **c.Open**$(m, r_c, c) \rightarrow (\{0,1\})$ takes in a message string, a commit randomness, and a commitment string and outputs 1 only if c is a valid commitment string of the tuple (m, r_c).

The algorithms **c.Commit**, **c.Open** satisfy the properties of a secure commitment scheme, where computational *binding* ensures that after committing to m_1, a Probabilistic Polynomial Time (PPT) adversary cannot find **c.Commit**(m_2, r_2)==**c.Commit**(m_1, r_1), with $(m_1, m_2) \in \mathcal{M}$, $(r_1, r_2) \in \mathcal{R}$, and $m_2 \neq m_1$. Further, anyone seeing c learns nothing on m due to the

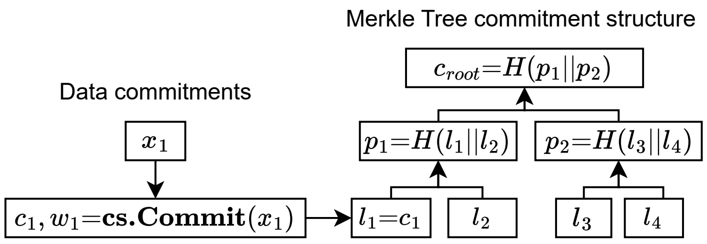


Figure 2.4.: Binary Merkle Tree commitment structure on a set of *data items* x_i, with $i \in \{0, \dots, N\}$. The depicted Merkle Tree has a depth D=2, leafs $l_1, \dots, l_{2^D}$, parents $p_1, p_{2*2^{D-1}-2}$, a root c_{root}, and depends on the hash function H. The root c_{root} represents the commitment string and the witness w consists of the internal witnesses w_i, with $i \in \{0, \dots, N\}$ and a Merkle path $f_{path}(x_i)$ that depends on the committed *data items*. In this figure, the witness comprises the set of tuples w=$[(w_1, [l_2^2, p_2^2] = f_{path}(x_1))]$, where l_2^2 indicates that l_2 is the second concatenation when computing p_1.

property of statistical *hiding*, where **c.Commit**(m_1, r_c) is statistically indistinguishable from **c.Commit**(m_2, r_c) with $(m_1, m_2) \in \mathcal{M}$ and $r_c \in \mathcal{R}$.

We use commitment schemes in combination with ZKP systems to construct enhanced privacy levels of *user* data. To do so, we leverage the fact that a commitment string can be opened privately if the ZKP circuit computes the **cs.Open** logic while taking the witness w as a private input. We rely on different algorithms to construct commitments (e.g. via hash functions or a Merkle Tree (MT) [49], [50]). For example, computing an MT inclusion proof against a commitment c_{root} requires the ZKP circuit to derive c_{root}' based on the secrets x_1 and w, and check if c_{root}'=c_{root} (cf. Figure 2.4).

Public Key Cryptography & Digital Signatures

PKC systems provide users with complementing key pairs to enable applications such as digital signatures or asymmetric encryption. The key property of PKC is that the keying material at users consists of a public (*public key*) and private (*private key*) part, where the private part is never disclosed. Using PKC, we define a digital signature scheme on a message string m with the algorithms, where

- **pk.Setup**$(1^\lambda) \rightarrow$ (*sk*, *pk*) uses a security parameter to output a PKC private key *sk* and public key *pk*.
- **pk.Sign**(*sk*, *m*) $\rightarrow$ (σ) takes as input the secret key and a message string *m*, and outputs the signature σ.
- **pk.Verify**(*pk*, *m*, σ) $\rightarrow \{0, 1\}$ takes as input the public key, the message message string, and a signature, and outputs either a 1 if the signature verification succeeds. Otherwise, the output is a 0.

A *user* calling **pk.Setup** and controlling the *private key sk* can sign a message m with **pk.Sign**, such that any *user* with the corresponding *public key pk* can verify the authenticity of m by calling **pk.Verify**. The PKC of the digital signature scheme guarantees that only the *user* with the *private key* can generate valid signatures according to the key pair.

Zero-knowledge Proof System

Proof systems allow a prover p to convince a verifier v of whether or not a statement is true. In theory, proof systems rely on an NP language $\mathcal{L}$ and the existence of an algorithm $R_{\mathcal{L}}$, which decides in polynomial time if w is a valid proof for the statement $x \in \mathcal{L}$ by evaluating $R_{\mathcal{L}}(x, w) \overset{?}{=} 1$. The assumption is that for any statement $x \in \mathcal{L}$, there exists a valid witness w and no witness exists for statements $x \notin \mathcal{L}$ [51], [52]. Proof systems provide the following properties:

Completeness which ensures that an honest prover convinces an honest verifier by presenting a valid witness for a statement.

Soundness which guarantees that a cheating prover cannot convince a honest verifier by presenting an invalid witness for a statement.

Zero-knowledge which guarantees that a malicious verifier does not learn anything except the validity of the statement.

HVZK which holds if the zero-knowledge property can be shown for a semi-honest verifier, who honestly follows the protocol definition.

ZKPs became prominent by securing cryptocurrencies to remain private but publicly verifiable [53]. Further, the field of verifiable credentials uses ZKPs to prove knowledge of signatures on data which is further asserted against public statements (e.g. age verification) [31], [54], [55]. Network policies rely on ZKPs to validate traffic compliance against block or allow lists [56], [57]. Data provenance protocols use ZKPs to validate integrity and non-ambiguity of TLS data [12], [58].

Zero-Knowledge Succinct Non-Interactive Argument of Knowledge: A zkSNARK proof system is a zero-knowledge proof system, where the four properties of *succinctness*, *non-interactivity*, computational sound *arguments*, and witness *knowledge* hold [59]. Succinctness guarantees that the proof system provides short proof sizes and fast verification times even for lengthy computations. If non-interactivity holds (cf. the Fiat-Shamir [60] heuristic in Appendix C), then the prover is able to convince the verifier by sending a single message. Computational sound arguments guarantee soundness in the zkSNARK system if provers are computationally bounded. Last, the knowledge property ensures that provers must know a witness to construct the proof.

We formally define a zkSNARK system with the following tuple of algorithms, where

- **zk.Setup**(1^{λ}, $ccs_{\mathcal{C}}$) $\rightarrow$ (pk, vk) takes as input a security parameter and a compiled constraint system expressing a circuit $\mathcal{C}$, and outputs the *prover* and *verifier* keys pk, vk.

- **zk.Prove**(ccs_C, w, pk) $\rightarrow \pi$ takes as input the compiled constraint system, a private witness, and the prover key pk and outputs a proof π.
- **zk.Verify**(w_{pub}, vk, π) $\rightarrow \{0, 1\}$ takes as input a public witness w_{pub}, the verifier key vk, and the proof π and outputs a 1 if π combined with vk successfully verify against w_{pub}. Otherwise a 0 is returned.

After running the **zk.Setup** algorithm, a *user* as the *prover* is able to compute a proof π by calling **zk.Prove** and share π with another *user* as a *verifier*. The proof π convinces the *verifier* of a statement expressed by a circuit C if the *verifier* successfully evaluates π using **zk.Verify**.

ZK Circuit Representation: ZKP systems rely on separate data representations to achieve the desired functionality. For instance, ZKP systems translate a constraint-based description of the computation into a provable arithmetic encoding via a frontend stack [61]. The arithmetic representation can be evaluated by the respective backend stack of the ZKP system. As of today, different domain-specific languages (DSLs) exist to describe the frontend, backend, or intermediate encodings of a computation circuit. Additional tooling exists to compile regular programs into ZKP circuits [62] or to provide interoperability between different ZKP encodings [63]. However, no tooling generates ZKP circuits at the abstraction level of policy compliance, which this thesis addresses.

Scope: In this thesis, we consider the *PlonK* Polynomial Interactive Oracle Proof (PIOP) system [52], [64], [65] to generate ZKPs in the form of zkSNARKs, which have the characteristics of succinct proof sizes and non-interactivity to convince a *verifier*. Similar to the work [66], we notice that identity systems introduce a setting where *provers* are exposed to limited computational resources (e.g. mobile phones) and *verifiers* have access to more powerful servers. Thus, we employ a proof system that uses a transparent Polynomial Commitment Scheme (PCS) (e.g. *FRI* [67], [68]), with quasilinear prover and polylogarithmic verifier complexities, for the off-chain computation and validation of ZKPs. To verify zkSNARKs on-chain, we rely on proof systems with a universal setup PCS (e.g. KZG [69]). The constant verification complexity of the universal setup PCS provides efficient verification of zkSNARKs at smart contracts [54].

ZKPs based on Garbled Circuits (GCs): Interestingly, zero-knowledge is a subset of secure 2PC and a ZKP can be computed using GC-based 2PC if only one party inputs private data. In this work, we make use of the HVZK notion based on boolean GCs [70]. In this setting, the garbler and constructor of the GC acts as the verifier and is assumed to behave semi-honest. The GC evaluates a public function f, which yields $\{0, 1\}$. The evaluator, as the prover, obtains the GC, input wire keys, and corresponding external labels but does not obtain the decoding table. After the prover evaluates the GC and returns the wire key which corresponds to a 1, the verifier is convinced of the proof. Formal security proofs for *completeness*, *soundness*, and HVZK of the garbled circuits proof system are provided in the work [70].

Table 2.1.: Notations and formulas of TLS variables.

Variable	Formula
H_2	H(ClientHello || ServerHello)
H_3	H(ClientHello || ... || ServerFinished)
H_6	H(ClientHello || ... || ServerCert)
H_7	H(ClientHello || ... || ServerCertVfy)
H_9	H(ClientHello || ... || ClientCertVfy)
label_{11}	"TLS 1.3, server CertificateVerify"
(k_{SATS}, iv_{SATS}) | (k_{CATS}, iv_{CATS})	DeriveTK(s=SATS|CATS) = (**hkdf.exp**(s,"key",H(" "),len(*k*)), **hkdf.exp**(s,"iv",H(" "),len(*iv*)))

Secret Sharing

Secret sharing involves a trusted dealer breaking a secret into shares with a **ss.Share** algorithm. Shares are distributed to qualified recipients which can reconstruct the secret by computing individual shares back together with a **ss.Reconstruct** algorithm [71]. In this work, we consider secret sharing with an access structure of t=n=2, where t out of n parties must add together secret shares to reconstruct the secret [72].

We formally define a secret sharing scheme with the following tuple of algorithms, where

- **ss.Setup**(λ) $\rightarrow$ (pp) takes in a security parameter and returns public parameters and randomness $r \overset{\$}{\leftarrow} \mathcal{R}(\lambda)$.
- **ss.Share**(pp, r) $\rightarrow$ ($\mathbf{r}$) takes in public parameters and randomness and returns additive secret shares $\mathbf{r}$=[r_1,..., r_n], where $\sum_{x=1}^{n} r_x = r$ holds.
- **ss.Reconstruct**($\mathbf{r}$) $\rightarrow$ (r) takes in additive secret shares and returns their sum.

2.2.3. Secure Protocols

For cryptographic protocols, we describe the inputs and outputs that are provided and obtained by involved parties. Additionally, we mention the security properties of exchanged parameters.

Transport Layer Security (TLS)

TLS is a standardized suite of cryptographic algorithms to establish secure and authenticated communication channels between two parties. TLS exists in different versions; TLS 1.2 and TLS 1.3. Generally, TLS has two phases, where the *handshake phase* derives cryptographic parameters to secure data sent in the *record phase*. TLS relies on the algorithms of Hash-based Message Authentication Code (HMAC) and HKDF to securely derive cryptographic parameters and relies on digital signatures to authenticate parties (cf. **ds.Sign**, **ds.Verify**, **hkdf.ext**,

```
TLS Handshake between the client $c$ and server $s$:
inputs: $x \xleftarrow{\$} \mathbb{F}_p$ by $c$. ($y \xleftarrow{\$} \mathbb{F}_p$, $sk_S$, $pk_S$) by $s$.
outputs: ($\mathrm{tk}_{\mathrm{CATS}}, \mathrm{iv}_{\mathrm{CATS}}, \mathrm{tk}_{\mathrm{SATS}}, \mathrm{iv}_{\mathrm{SATS}}$) to $c$ and $s$.
1. $c$: $X = x \cdot G$; send $X$ in $m_{\mathrm{CH}}$
2. $s$: $Y = y \cdot G$; send $Y$ in $m_{\mathrm{SH}}$
3. $b$: dES = **hkdf.exp**(**hkdf.ext**(0,0),“derived” || H(“ ”))
4. $b$: DHE = $x \cdot y \cdot G$; HS = **hkdf.ext**(dES, DHE)
5. $b$: SHTS = **hkdf.exp**(HS,“s hs traffic” || $\mathrm{H}_2$)
6. $b$: CHTS = **hkdf.exp**(HS,“c hs traffic” || $\mathrm{H}_2$)
7. $b$: $(\mathrm{k}_{\mathrm{CHTS}}, \mathrm{iv}_{\mathrm{CHTS}})$ = DeriveTK(CHTS)
8. $b$: $(\mathrm{k}_{\mathrm{SHTS}}, \mathrm{iv}_{\mathrm{SHTS}})$ = DeriveTK(SHTS)
9. $b$: $\mathrm{fk}_S$ = **hkdf.exp**(SHTS, “finished” || “ ”)
10. $s$: SCV=**ds.Sign**($sk_S$,$\mathrm{label}_{11}$||$\mathrm{H}_6$); send SCV in $m_{\mathrm{SCV}}$
11. $s$: SF = **hmac**($\mathrm{fk}_S$, $\mathrm{H}_7$); send SF in $m_{\mathrm{SF}}$
12. $c$: SF′ = **hmac**($\mathrm{fk}_S$, $\mathrm{H}_7$); verify SF′$\overset{?}{=}$ SF
13. $c$: **ds.Verify**($pk_S$, $\mathrm{label}_{11}$ || $\mathrm{H}_6$, SCV) $\overset{?}{=} 1$
14. $b$: $\mathrm{fk}_C$ = **hkdf.exp**(CHTS, “finished” || “ ”)
15. $c$: CF = **hmac**($\mathrm{fk}_C$, $\mathrm{H}_9$); send CF in $m_{\mathrm{CF}}$
16. $s$: CF′ = **hmac**($\mathrm{fk}_C$, $\mathrm{H}_9$); verify CF′$\overset{?}{=}$ CF
17. $b$: dHS = **hkdf.exp**(HS,“derived” || H(“ ”))
18. $b$: MS = **hkdf.ext**(dHS, 0)
19. $b$: CATS = **hkdf.exp**(MS, “c ap traffic” || $\mathrm{H}_3$)
20. $b$: SATS = **hkdf.exp**(MS, “s ap traffic” || $\mathrm{H}_3$)
21. $b$: $(\mathrm{k}_{\mathrm{CATS}}, \mathrm{iv}_{\mathrm{CATS}})$ = DeriveTK(CATS)
22. $b$: $(\mathrm{k}_{\mathrm{SATS}}, \mathrm{iv}_{\mathrm{SATS}})$ = DeriveTK(SATS)
```

Figure 2.5.: TLS 1.3 specification of session parameters. Characters at the beginning of lines indicate if the server s, the client c, or both parties b call the functions per line.

hkdf.exp, **hmac** in Figure 2.5). We provide further details of TLS-specific security algorithms in Section 2.2.2 and present TLS-specific transcript hashes, labels, and key derivation functions of traffic keys in Table 2.1.

Handshake Phase:

1. **Key Exchange and Key Derivation:** To establish a secure channel between a server and a client, TLS relies on the Diffie-Hellman Key Exchange (DHKE) to securely exchange cryptographic secrets between two parties (cf. Figure 2.5, lines 1-4). For example, with TLS 1.3 configured to use elliptic curve cryptography, parties protect secrets x, y with an encrypted representation X, Y and exchange X, Y via the Client Hello (CH) and Server Hello (SH) messages m_{CH}, m_{SH}. With access to X, Y, only the

client and server can securely derive the Diffie–Hellman Exchange (DHE) key, where DHE $= x \cdot y \cdot G = y \cdot X = x \cdot Y$ holds. Both parties continue to use DHE to derive traffic secrets. In the TLS 1.3 One Round-trip Time (1-RTT) mode, the *server* is able to encrypt all server-side handshake messages after receiving a supported client key share in the CH message m_{CH}.

In contrast, TLS 1.2 exchanges the messages m_{CH}, m_{SH} in plain and refers to the DHE value as the premaster secret. TLS 1.2 uses the premaster secret together with the client and server randomness to derive a master secret, which, in turn, is used to derive traffic secrets. When TLS 1.2 is configured to use AEAD based on stream ciphers, TLS 1.2 generates two application traffic keys to secure record phase traffic (k_{CATS}, k_{SATS}). Otherwise, if TLS 1.2 uses a Cipher Block Chaining (CBC) mode to encrypt records, TLS 1.2 generates additional Message Authentication Code (vMAC) keys. In contrast to the GCM mode, the CBC mode counts as key-committing [56], [57], which guarantees the existence of a non-ambiguous mapping between traffic secrets and authentication tags. Per default, TLS 1.3 generates two keys (k_{CHTS}, k_{SHTS}) to secure handshake traffic and generates two keys to secure record traffic (k_{SATS}, k_{CATS}). Due to the key-independence property of TLS 1.3 [73], disclosing handshake traffic secrets (e.g. Server Handshake Traffic Secret (SHTS)) does not compromise the security of record traffic secrets. For instance, to compute the Server Application Traffic Secret (SATS), a party requires access to the Handshake Secret (HS). Even though, HS is used to derive SHTS (cf. line 5 of Figure 2.5), **hkdf.exp** prevents the reconstruction of HS from SHTS.

2. **Authenticity:** To mutually authenticate each other, both parties exchange certificates and compute authentication parameters (cf. Figure 2.5, lines 9-16). Notice that in TLS, client-side authentication is optional, which is why we omit client certificates in Figure 2.5. But, we show the computations of the Server Finished (SF) and Client Finished (CF) values, because, to constitute an authenticated TLS session, both parties must successfully exchange and verify the SF and CF messages $m_{\text{SF}}, m_{\text{CF}}$. For server-side authentication, the server computes the certificate verification value (e.g. SCV), which binds a PKI X.509 certificate to the TLS transcript via a digital signature [74]. Here, the signature is computed with the server secret key sk_S and is verified with the corresponding server public key pk_S. The client obtains the server public key pk_S in the PKI certificate and aborts the TLS session if the signature verification fails.

Record Phase: The TLS record phase requires parties to protect data with Authenticated Encryption (AE) algorithms before data can be exchanged. AE algorithms translate plaintext data **pt** into a confidential and authenticated representation ($\mathbf{ct}, t$), with ciphertext **ct** and authentication tag t. Ciphertext data is computed based on block or stream cipher algorithms and depends on keys established in the handshake phase. We elaborate on TLS data protection algorithms in the Appendix 2.2.2.

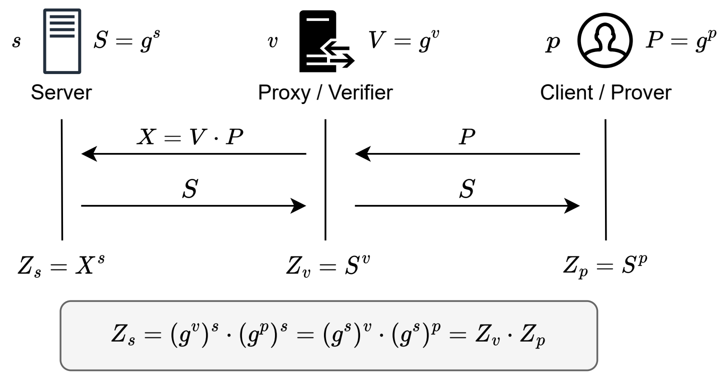


Figure 2.6.: Illustration of the Three-party Handshake (3PHS) and exchanged cryptographic parameters between the server, the proxy, and the client. The gray box at the bottom indicates the relationship between shared client-side secrets Z_v and Z_p, which corresponds to the session secret Z_s of the server.

Three-party Handshake

In the 3PHS (cf. Figure 2.6), each party picks a secret randomness (s, v, p) and computes its encrypted representation (S, V, P). By sharing $V + P = X$ with the server in the CH, the server derives the session secret $Z_s = s \cdot X$, which corresponds to the TLS 1.3 secret DHE. When the server shares S in the SH, both the proxy and client derive their shared session secrets Z_v and Z_p respectively such that $Z_s = Z_v + Z_p$ holds. In the end, neither the client nor the verifier has full access to the DHE secret of the TLS handshake phase. The 3PHS works for both TLS versions but in Figure 2.6, we show a TLS 1.3-specific configuration based on the Clliptic Curve Diffie–Hellman Exchange (ECDHE), where the parameters (e.g. Z_p) are EC points structured as $P = (x, y)$.

Elliptic Curve to Finite Field (EC2F) Conversion

The ECTF algorithm is a secure 2PC protocol and converts multiplicative shares of two EC x-coordinates into additive shares [11], [12]. Figure 2.7 shows the computation sequence of the ECTF protocol which makes use of the vector MtA algorithm defined in Section 2.2.2. By running the ECTF protocol, two parties p_1 and p_2, with EC points P1, P2 as respective private inputs, mutually obtain additive shares s_1 and s_2, which sum to the x-coordinate of the EC points sum P1+P2. TLS oracles use the ECTF protocol to transform the client-side EC secret shares Z_v and Z_p into additive shares s_v and s_p [11], [12]. Since the relation $s_v + s_p = x$ for $(x, y) = Z_s$ holds, it becomes possible to follow the TLS specification by using secure 2PC based on boolean garbled circuits with bitwise additive shares as input.

Semi-honest 2PC

Secure 2PC allows two mutually distrusting parties with private inputs x, y to jointly compute a public function $f(x, y)$ without learning the counterparty's private input [75], [76]. A 2PC

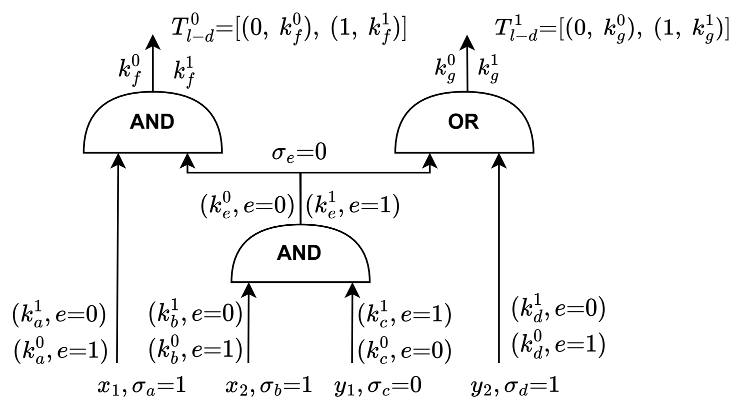
```
**ECTF** between two parties $p_1$ and $p_2$.
**inputs:** $P_1 = (x_1, y_1)$ by $p_1$, $P_2 = (x_2, y_2)$ by $p_2$.
**outputs:** $s_1$ to $p_1$, $s_2$ to $p_2$.
$p_1$: $(sk,pk)$=**mta.Setup**$(1^\lambda)$;   send $pk$ to $p_2$
$p_1$: $\rho_1 \xleftarrow{\$} \mathbb{Z}_p$;   **c1**=**mta.Enc**$([-x_1,\rho_1], sk)$;   send **c1** to $p_2$
$p_2$: $\rho_2 \xleftarrow{\$} \mathbb{Z}_p$;   $(c2,\beta)$=**mta.Eval**$(\mathbf{c1},[\rho_2,x_2],pk)$;   $\delta_2=x_2 \cdot \rho_2+\beta$;   send $(c2,\delta_2)$ to $p_1$
$p_1$: $\alpha$=**mta.Dec**$(c2,sk)$;   $\delta_1=-x_1 \cdot \rho_1+\alpha$;   $\delta=\delta_1+\delta_2$;   $\eta_1=\rho_1 \cdot \delta^{-1}$;
    **c1**=**mta.Enc**$([-y_1,\eta_1],sk)$;   send $(\mathbf{c1},\delta_1)$ to $p_2$
$p_2$: $\delta=\delta_1+\delta_2$;   $\eta_2=\rho_2 \cdot \delta^{-1}$;   $(c2, \beta)$=**mta.Eval**$(\mathbf{c1},[\eta_2,y_2],pk)$;   $\lambda_2=y_2 \cdot \eta_2+\beta$;
    send c2 to $p_1$
$p_1$: $\alpha$=**mta.Dec**$(c2,sk)$;   $\lambda_1=-y_1 \cdot \eta_1 + \alpha$;   **c1**=**mta.Enc**$([\lambda_1],sk)$;   send **c1** to $p_2$
$p_2$: $(c2,\beta)$=**mta.Eval**$(\mathbf{c1}, [\lambda_2], pk)$;   $s_2 = 2 \cdot \beta + \lambda_2^2 - x_2$;   send c2 to $p_1$
$p_1$: $\alpha$=**mta.Dec**$(c2,sk)$;   $s_1 = 2 \cdot \alpha + \lambda_1^2 - x_1$
```

Figure 2.7.: The Elliptic Curve to Field (ECTF) algorithm converts multiplicative shares in form of EC point x-coordinates from points $P_1, P_2 \in EC(\mathbb{F}_p)$ to additive shares $s_1, s_2 \in \mathbb{F}_p$. It holds that $s_1 + s_2 = x$, where x is the coordinate of the EC point $P_1 + P_2$.

system based on boolean garbled circuits involves a party p_1 with input $\mathbf{x}$ as the garbler and party p_2 with input $\mathbf{y}$ as the evaluator. Party p_1 is supposed to generate the garbled circuit $\boldsymbol{G}(\mathcal{C})$, where the boolean circuit $\mathcal{C}$ implements the logic of the public function f (cf. Figure 2.8). To generate the garbled circuit, p_1 randomly samples wire keys $\mathbf{k}_L^0, \mathbf{k}_L^1$ and a signal bit σ_L at every wire w_L. For the purpose of evaluating the function f, wire keys $\mathbf{k}_L^{\mathbf{i}}$ encode binary data representations of f using internal labels $\mathbf{i}$. The purpose of signal bits is twofold. Signal bits encrypt internal bits to external bits e_L=$\sigma \oplus \mathbf{i}$ which can be shared with p_2. With that, signal bits enable the evaluator to discover valid entries of garbled tables $G(\mathcal{C})$ through external bits e [77]. Further, signal bits randomize garbled truth tables $G(\mathcal{C})$ to obfuscate truth table bit mappings.

Once wire keys, signal bits, and external labels exist, p_1 computes the garbled table entries as follows. Per row of table $G(\mathcal{C})$ (cf. Figure 2.8), the bit tuples in the left column are combinations of external labels that correspond to incoming gate wires. The right column contains double encrypted wire keys that correspond to outgoing gate wires. For gates yielding output labels, garbled entries encrypt wire keys. For intermediate gates, garbled entries encrypt wire keys concatenated with corresponding external labels.

After garbling a circuit, p_1 shares $\boldsymbol{G}(\mathcal{C})$, $\boldsymbol{T}_{l-d}$, and, if $\mathbf{x}$=[1,0], $(k_a^1, e$=0) and $(k_b^0, e$=1) with p_2. To obtain wire keys that correspond to the input bits of $\mathbf{y}$, p_2 interacts with p_1 in two 1-out-of-2 OT protocol executions. The OT_2^1 scheme involves two parties where party p_1 sends two messages m_1, m_2 to party p_2 and does not learn which of the two messages m_b is revealed to party p_2. Party p_2 inputs a secret bit b which decides the selection of the message m_b. In this work, we make use of the OT_2^1 scheme defined in the work [48], which does not require a



$G(\mathcal{C}^{0,(1,2)}_{\text{AND}})$	
0,0	$\mathrm{E}_{k^1_b}(\mathrm{E}_{k^0_c}(k^0_e\|\|0))$
0,1	$\mathrm{E}_{k^1_b}(\mathrm{E}_{k^1_c}(k^1_e\|\|1))$
1,0	$\mathrm{E}_{k^0_b}(\mathrm{E}_{k^0_c}(k^0_e\|\|0))$
1,1	$\mathrm{E}_{k^0_b}(\mathrm{E}_{k^1_c}(k^0_e\|\|0))$

$G(\mathcal{C}^{1,(0,1)}_{\text{AND}})$	
0,0	$\mathrm{E}_{k^1_a}(\mathrm{E}_{k^0_e}(k^0_f))$
0,1	$\mathrm{E}_{k^1_a}(\mathrm{E}_{k^1_e}(k^1_f))$
1,0	$\mathrm{E}_{k^0_a}(\mathrm{E}_{k^0_e}(k^0_f))$
1,1	$\mathrm{E}_{k^0_a}(\mathrm{E}_{k^1_e}(k^0_f))$

$G(\mathcal{C}^{1,(1,3)}_{\text{OR}})$	
0,0	$\mathrm{E}_{k^0_e}(\mathrm{E}_{k^1_d}(k^1_g))$
0,1	$\mathrm{E}_{k^0_e}(\mathrm{E}_{k^0_d}(k^0_g))$
1,0	$\mathrm{E}_{k^1_e}(\mathrm{E}_{k^1_d}(k^1_g))$
1,1	$\mathrm{E}_{k^1_e}(\mathrm{E}_{k^0_d}(k^1_g))$

Figure 2.8.: Example of a garbled circuit $\mathcal{C}$ expressing the function f of a secure computation via boolean logic gates. Every circuit wire w_L is encoded with secret internal labels $\mathbf{i}$, a secret and random signal bit σ_L, external labels $\mathbf{e}$=$\sigma_L \oplus \mathbf{i}$ (where $i, e, \sigma \in \{0, 1\}$), and wire keys $\mathbf{k}^{\mathbf{i}}_L$. Internal labels are associated with input data bits and the lists $\boldsymbol{T}_{l-d}$ map output labels to output data bits. The gate-wise garbling tables $\boldsymbol{G}(\mathcal{C})$ map tuples of external labels to garbled labels concatenated with external labels. Output wires have neither external labels nor signal bits.

trusted setup. The trusted setup procedure introduces a third party that (i) takes over the generation of cryptographic material and (ii) is trusted to delete the underlying random parameters of the material.

The OT protocol requires p_1 to share $k^{\mathbf{y}}_e, k^{\mathbf{y}}_d$ with corresponding external values with p_2. Further, the OT scheme gives p_2 access to the keys $(k^0_c, e{=}0)$ and $(k^1_d, e{=}0)$ if $\mathbf{y}$=[0,1], and prevents p_1 from learning p_2's selection of wire keys. With access to $\boldsymbol{G}(\mathcal{C})$, input wire keys, and corresponding external labels, p_2 is able to evaluate the garbled circuit. To evaluate the first output bit, p_2 decrypts the third entry of table $G(\mathcal{C}^{0,(1,2)}_{AND})$ and obtains $(k^0_e, e{=}0)$. With that, p_2 continues to decrypt the first entry of table $G(\mathcal{C}^{1,(0,1)}_{AND})$ to obtain k^0_f (cf. Figure 2.8). Last, p_2 decodes k^0_f using the decoding table T^0_{l-d} to obtain the first output bit 0. If required, p_2 shares the obtained 2PC output back to p_1.

Maliciously Secure 2PC

Multiple approaches have been proposed to construct maliciously secure 2PC.

Cut-and-Choose Paradigm: The cut-and-choose paradigm requires the garbler to prepare n garbled circuits out of which the evaluator randomly picks one. After the random choice of the evaluator, the garbler is supposed to share the parameterization of all garbled circuits except for the one chosen by the evaluator. With all circuit parameters, the evaluator can check the correctness of the obtained garbled circuits and abort if any failure is detected. If the check at the evaluator succeeds, then both parties continue to use the circuit chosen by the evaluator to proceed with the semi-honest 2PC protocol [78]. The cut-and-choose paradigm is expensive with regard to bandwidth requirements and gives a malicious garbler a $\frac{1}{n}$ chance of being caught.

Authenticated Garbling Paradigm: The concept of authenticated garbling adds information-theoretic MACs to the entries of garbled tables. As a result, the garbled tables are secretly shared between both parties via the information-theoretic authentication bits. After garbling the circuit, the garbler sends her shares of the garbled tables to the evaluator. The evaluator is able to authenticate each masked bit that it learns. Due to the fact that the 2PC function is public, the evaluator is able to check the circuit correctness by evaluating the information-theoretic MACs according to the public function [79]. The concept of authenticated garbling is the most efficient approach to achieve maliciously secure 2PC. However, due to the fact that our optimizations mostly rely on semi-honest 2PC systems, we consider another approach of maliciously secure 2PC.

Dual-execution Paradigm: We consider the work [80] to secure semi-honest 2PC against malicious adversaries. The 2PC dual-execution protocol runs two instances of the semi-honest 2PC, where both parties p_1 and p_2 successively act as the garbler and evaluator [80]. Before any 2PC output is shared with the counterparty, the protocol runs a secure validation phase on obtained outputs. The validation phase checks if both semi-honest 2PC executions yield the same output.

Secure 2PC using the dual-execution mode is susceptible to a single-bit leakage. However, if the input is random and leaking even the most significant bit does not yield any advantage, then using this technique is sufficient. The key derivation functions in TLS oracles (e.g. 2PC SHTS circuit) depend on sufficiently random inputs with secret shared input parameters. Thus, applying the dual-execution mode in TLS oracles is feasible [81].

Intuition of the Dual-execution Mode: The idea of the mutual output verification is as follows. If p_1, as the evaluator, obtains output wire keys $\mathbf{k}_x$ and output bits $\mathbf{b}$ from a correctly garbled circuit of p_2, then p_1 knows which output labels $\mathbf{k}_y$ according to $\mathbf{b}$ p_2 must evaluate on a correctly garbled circuit of p_1. Thus, if p_1 shares a commitment in form of a hash $H(\mathbf{k}_y||\mathbf{k}_x)$ with p_2 after the first circuit evaluation, and p_2 returns the same hash $H(\mathbf{k}_y||\mathbf{k}_x)$ after the second circuit evaluation, then p_1 is convinced of a correct garbling by p_2. Because, if p_2 incorrectly garbles a circuit, then p_1 obtains the bits $\mathbf{b'}$. And, if p_1 correctly garbles a circuit, p_2 obtains correct bits $\mathbf{b}$. The incorrect bits $\mathbf{b'}$ lead p_1 to a selection of labels $\mathbf{k'}_x$ and $\mathbf{k'}_y$ and the correct bits $\mathbf{b}$ lead p_2 to a correct selection of $\mathbf{k}_y \neq \mathbf{k'}_y$. Since p_2 does not know that output keys p_1 evaluates, p_2 cannot predict any keys $\mathbf{k'}_x, \mathbf{k'}_y$ which lead to the hash that is expected by p_1. To communicate the output of a maliciously secure 2PC to a single party, only the first garbler is required to share the output decoding table with the counterparty.

3. Sovereign Identity & Data Privacy

In order to solve the existing misalignment between data protection and consent, we re-assess online identity under consideration of the latest technological developments in the field of data privacy and security. To address the emerging requirements and liabilities of data protection, we explore the interplay of PETs and decentralized public infrastructures. Further, we seek to equip data controllers with strong data sovereignty capabilities at all times. This involves the right and power to publish the policy-driven consent or revocation of any digital agreements. As the core enabler of sovereign online identity, we rely on persistent access to cryptography at the data controller itself. Such that at least a share of custody remains connected to an accountable online identity. The first two contributions of this work follow the above-stated principles and facilitate the expression of consent on protected digital data.

In more detail, we introduce new approaches to managing digital identifiers and associated data in the following sections. Our approaches equip end devices with new data sovereignty and privacy guarantees and improve the interplay between PETs and decentralized public infrastructures. We divide our contributions [82] and [83] into two sections. In Section 3.2, we investigate the secure verification of ZKPs at smart contracts. Based on our findings, we propose a new identity system, called *Portal*, where end devices maintain the authority of private data administration. In Section 3.3, we investigate privacy-preserving authorization techniques enabled by cryptographic accumulators. We apply our findings in the context of data certification systems and secure the policy-driven attestation of private data. Before we present our contributions, we outline important insights and challenges which guide our research. Our insights and challenges are the result of another contribution [84] which systematizes the scope of data sovereignty that is achievable today.

3.1. Systematization of Data Sovereignty

In this work, the definition of data sovereignty concerns the notion of how much control over data a device can obtain and maintain when participating online. This involves the administration of identifiers and associated data, access control to data, and the outsourced processing of data. Data sovereignty can also be defined with regard to the geographical location of data [85]. Here, the sovereignty of data is determined by the geopolitical laws that apply to the region where the data is managed. To exclude any danger of confusion, this work considers the technical definition of data sovereignty proposed by our contribution [84], where sovereignty is defined as a notion of control via advanced cryptographic techniques.

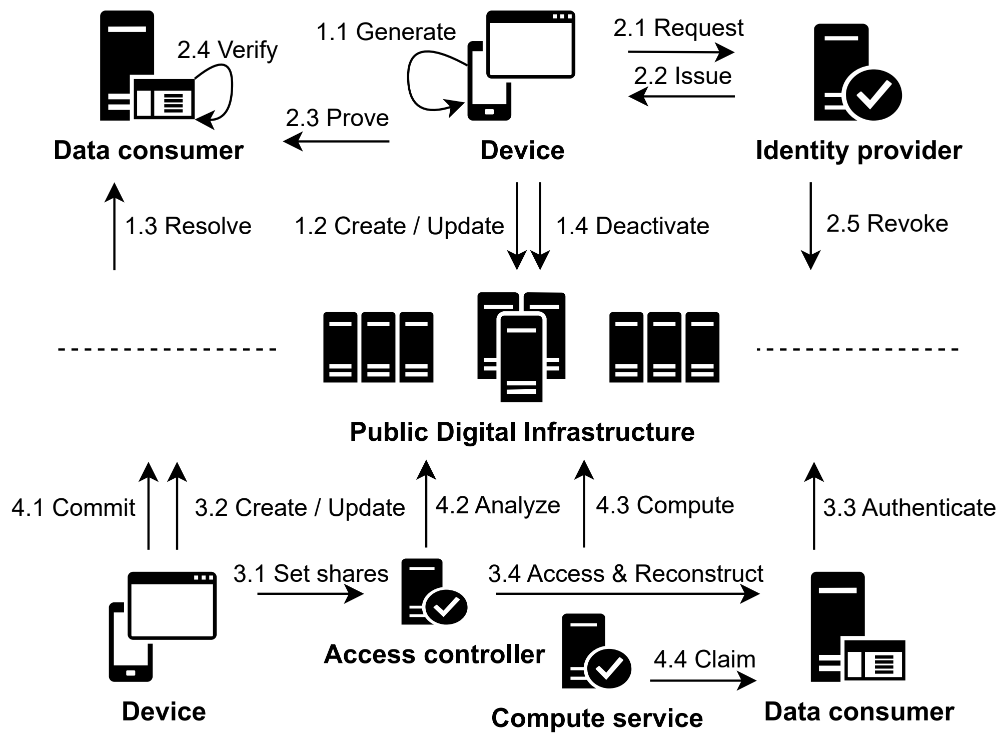


Figure 3.1.: The scope of data sovereignty with respect to the different fields of data management. The process 1.X describes necessary functionalities to create and maintain identifiers. The process 2.X indicates the functionalities of a certification system which issues, presents, and verifies credentials. The process 3.X defines the functionalities of accountable access control. The process 4.X achieves the strongest notion of data sovereignty and defines the policy-driven processing of data that is never disclosed.

3.1.1. Scope of Sovereignty

In terms of the technical notion, data sovereignty manifests itself in different forms depending on the scope and type of the data that is managed (cf. Figure 3.1). According to the scope, data sovereignty covers the fields of identifier management, data certification in the form of credentials, access control to data, and the policy-compliant processing of data [84]. Depending on the type of data, data sovereignty concerns the accountable control, protection, and consent over data. No explicit data protection is required if devices manage one-time public identifiers. However, in this case, data sovereignty must ensure a persistent notion of control over the digital identifier. If devices manage data attributes and credentials, data sovereignty concerns the protection of access during data sharing and outsourced processing of data. In the following, we summarize how much data sovereignty can be achieved today using advanced cryptographic techniques.

Identifier & Data Management

In traditional approaches of identity management, third-party service providers or intermediaries take over the custody to manage identifiers and data on behalf of devices. In this scenario, devices have little sovereignty over how their data is being shared and processed. The sovereignty over identifiers and data increases in the setting of decentralized public infrastructures, where intermediary services are replaced by autonomous compute programs such as smart contracts. The decentralized setting gives devices access to cryptographic key pairs, which devices generate themselves (cf. step 1.1 of Figure 3.1). With access to key pairs, devices are capable of answering key ownership challenges, which uniquely determine the holder of a key pair. If keys are secret-shared between multiple parties, then devices must collaboratively answer key ownership challenges. Notice that devices have the power to decide if any collaborative ownership setting should be established. Thus, devices are empowered to determine if identifiers or data should be administered individually or with shared custody. Further, key pairs can be used to authenticate data via digital signatures, which unequivocally bind data to the key holders. Signatures authenticate the transaction-based computing in decentralized public infrastructures. Hence, key holders have the opportunity to create, update, or deactivate digital agreements transparently (cf. steps 1.2-1.4 in Figure 3.1). The immutability and availability guarantees of the public infrastructure ensure that devices remain accountable at all times.

Policy-driven Data Attestation

To increase the trust in data attributes that are associated with an identifier, data holders typically request and obtain attestations in the form of credentials (cf. steps 2.1-2.2 in Figure 3.1). Credentials are issued by services that witness the online activities of data holders. Otherwise, credentials are issued once certain facts on the data have been verified successfully. The verified statement is typically carried by the credential itself and counts as a policy. Online services assume that the data complies with the policy expressed by the credential.

If devices present credentials to data consumers (e.g. web services), then consumers audit the credentials via a certification chain. The certification chain is a collection of connected credentials that lead the consumer to a root attestation. The root attestation must be trusted by the consumer. Further, if consumers do not trust the certification system, then consumers abort the acceptance of credential presentations. In the traditional web, credentials are issued to web services or intermediate end-points. Only on rare occasions, end-devices obtain access to credentials (e.g. S/MIME email certificates at client programs). The idea of sovereign identity reconsiders credential management and brings credential management down to end devices. As such, users as credential holders have the power to decide if and to whom credentials should be presented (cf. step 2.3 in Figure 3.1). Decentralized credentials are connected to a key pair of the holder such that the key ownership challenge additionally determines credential ownership. Consumers are supposed to resolve certificate chains via publicly accessible infrastructures. Equally, as in the traditional web, attesting parties maintain the right to revoke credentials.

To enhance the sovereignty of data privacy in decentralized certification systems, recent approaches have constructed anonymous credential schemes, which are powered by ZKP technology and group signature schemes [40], [86].

Policy-driven Access Control

If identifiers, data, and credentials reside at end devices, then, as a natural consequence, the question emerges if and how the administration of access control mechanisms can be shifted to the end device. In order to maintain strong sovereignty guarantees, access control policies should reside in the hands of the data controllers. In addition, the data holder should be capable of verifying the accountability of access-seeking consumers before access is granted. Otherwise, consumers have the potential to circumvent promises made before receiving access.

The system design to achieve decentralized access control with strong data sovereignty guarantees is depicted via the third process in Figure 3.1. Here, data holders use a symmetric encryption key to protect data. Protected data can be safely stored externally. Next, the key is threshold secret shared with a committee of access controllers due to the fact that current blockchains have no support for secret management [87]. Subsequently, devices upload an access policy to the public infrastructure. Once the access policy is publicly available, consumers have to post policy-compliant transactions to the public infrastructure. This way, the data holder is empowered to determine how consumers authenticate for data access. Typically, the access policy sets a timestamp which upon expiry allows consumers to claim access from the committee of access controllers. This mechanism ensures that consumers commit to agreements made in the access policy before access is granted. If consumers claim access, then the secret management committee audits the public infrastructure for the policy-compliant transactions of requesting users. If consumers can be associated with compliant access transactions, then individual parties of the committee release secret shares of the symmetric encryption key to the consumer. With access to a threshold number of secret shares, consumers reconstruct the encryption key. Since the access policy conveys the storage location of protected data, consumers are able to download protected data and decrypt it with the symmetric key [88].

Policy-driven Data Processing

Concerning data sovereignty, every access control scheme faces a common problem. Data counts, by its nature, as non-rival. This means that data can be easily copied once access has been obtained. If a copy exists, the rivalry of creating the same data instance is lost. Data non-rivalry is problematic because the original data holder cannot impose or enforce any ruling on how a shared data instance should be treated. This opens the door for undetected, unintentional use of shared data by consumers.

The only way to solve the problem of data non-rivalry is to prevent access to the data instance x itself. Instead, consumers can be granted access to a function output $f(x)$ which processes the data instance x. It must hold that the consumer never learns any information on the inputs x of the function f. This requirement can be ensured if the function f is evaluated using secure computation techniques (cf. Section 2.2.3). Further, it must hold that the function evaluation itself does not leak any information to the consumer (e.g. through side channels). The security technique of differential privacy guarantees that the execution of an algorithm leaks less than an ϵ-threshold of information to the consumer [89].

To enable policy-driven data processing, devices need the capability to verify the compliant data processing according to a policy (cf. process 4.X in Figure 3.1). Here, the policy defines a sequence of transactions that must appear before the consumer is granted access to a function output. As such, the access committee must audit the compute program on the support of differential privacy and the secure implementation of security algorithms that protect the inputs. Further, if a compliant program is executed, the compute node is required to post a proof of correct function evaluation to the public infrastructure. If all requirements defined by the policy are met, the access control committee atomically releases the function output to the consumer [90].

3.1.2. Insights & Challenges

Even though constructing a strong notion of data sovereignty is possible using advanced cryptographic techniques, it remains to investigate the practicability and tradeoffs between different technical solutions. Further, the question remains how much sovereignty a user or device needs in order to perform existing online functionalities. We define key insights and challenges of sovereign identity and credential management next with the goal of solving identified issues with our contributions.

As a first insight, we notice that the construction of strong sovereignty notions does not align with existing online functionalities. For instance, it remains a challenge to design a complete sign-on system in the context of decentralized public infrastructures with equal capabilities as dominating solutions such as OAuth 2.0 or *OpenID Connect*. At the same time, it remains a challenge to integrate enhanced data sovereignty and privacy concepts. Because, until today, most decentralized public infrastructures lack native support for verifying PET technologies such as ZKPs [25]. Further, PET technologies must be adapted or potentially advanced to reflect the self-determined control of an individual or a collective.

As a second insight, we notice that the emerging field of decentralized computation and data management yields plenty of novel approaches. For example, the W3C actively standardizes the concept of decentralized identifiers [29] and verifiable credentials [31] in the context of public infrastructures. However, it remains unclear if the standardized concepts match with real-world requirements that are imposed by decentralized public infrastructures. We address this challenge by investigating if and how novel proposals respect sovereignty guarantees for all involved stakeholders.

The following sections explain how our research contributions solve the core challenges of data consent and protection in the decentralized context. Our contributions address the above-stated challenges and support the self-determined protection of digital data during online interactions.

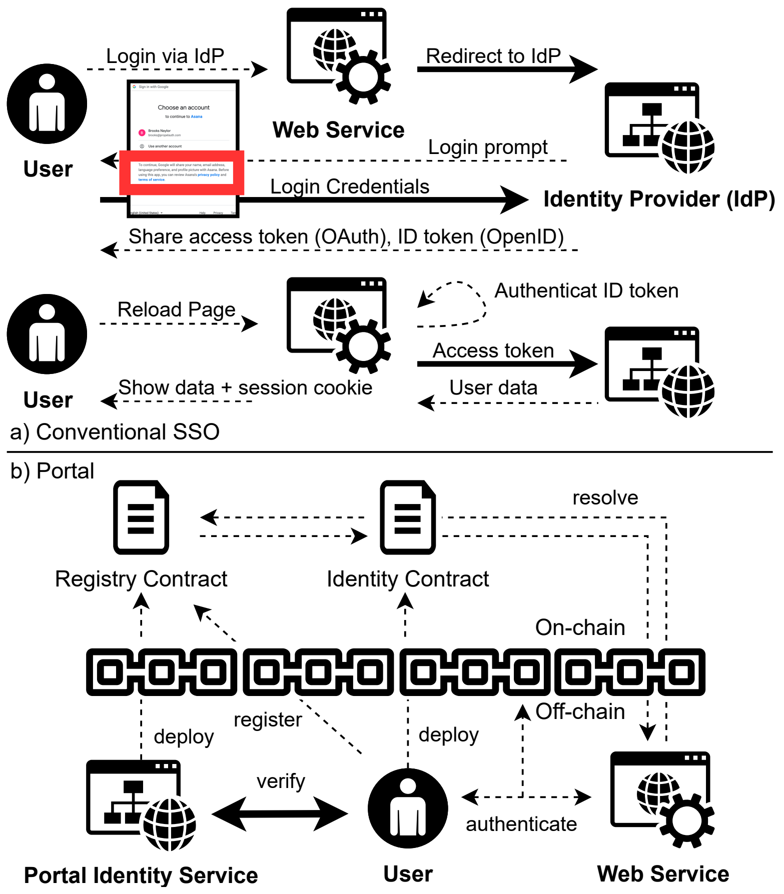


Figure 3.2.: a) Overview of the Single Sign-On (SSO) delegated authentication and authorization where the *user* agrees to a fixed policy (red box) of the Identity Provider (IdP). Bold arrows indicate *user*-to-IdP interactions which track *user* activities. b) Simplified view of the *Portal* identity system, where *users* manage data and authenticate towards web services with self custody. © 2024 IEEE

3.2. Time-bound and Replay-resistant Proofs on Public Blockchains

As initially stated, the latest identity systems rely on public blockchains to enhance user autonomy and reduce tracking from conventional identity providers. At the same time, identity systems integrate novel technologies such as zero-knowledge proofs (ZKPs) to improve data privacy and data compliance. We show that a naive verification of ZKPs at smart contracts enables replay attacks: Attackers can replay ZKPs at arbitrary times without having access to the private inputs that are required for the computation of the ZKP. To solve this problem, we construct a transaction sequence that verifies time-bound and replay-resistant ZKPs at smart contracts. Our construction introduces an additional but constant fee of 0.14$ per verification of a ZKP on the public blockchain Ethereum. With our new construction, we propose Portal, a novel identity system for decentralized single sign-on.

Motivation: Almost every service of today's web manages *users* based on an identifiable session and requires a mechanism to authenticate *users* beforehand. The *user* authentication uniquely identifies every *user* of the system and guarantees that the session is unique to one *user*. To prevent each web service from implementing its own identity and authentication system, *OpenID Connect*, as the latest SSO protocol, was standardized in 2014 [21]. The SSO paradigm delegates *user* authentication at a web service towards a third-party IdP, which handles the unique identification of the *user*. Once the *user* logs in at the IdP, the IdP shares a token with the *user*, which, upon reloading the connection to the web service, is shared with and used by the web service to authenticate the *user* (cf. top half of Figure 3.2). *OpenID Connect* is an authentication layer in the OAuth 2.0 protocol, which, beyond *user* authentication, delegates access control of *user* resources via an authorization server. It is common that the same organization running the IdP operates the authorization server [91]. In OAuth 2.0, the *user* can agree to a fixed policy of the IdP which determines the access privilege of the web service towards *user* data.

Even though delegated authentication and authorization via SSO is very convenient and cheap for *users* and web services, IDPs gain the option to track every login and data access of a *user*. Further, since IDPs share fixed policies, which *users* must agree or disagree with, *users* lack control and responsibility over deciding which and how data is shared with the web service. To solve the misaligned incentives between all parties, recent approaches such as SIWE [27] leverage new technologies and replace the IdP with a public blockchain. Blockchains return data responsibility back to *users* and distribute the single trusted IdP over an open network of trustless validators. Blockchain validators securely update an agreed-upon global state of accounts by verifying transactional state updates which are controlled by *users*. Thus, the global state of accounts depends on externally-issued transactions that are owned by *users*, and, with that, provide *users* with new notions of data sovereignty, self-custody, and autonomy [84]. Beyond SIWE, novel privacy-enhancing technologies such as ZKP systems led to a reconsideration to standardize identity systems [29]–[31]. For instance, Polygon ID [92] employs ZKP technology to enhance the data privacy and data compliance of *users*. Further, modern identity systems rely on certification ecosystems, where issuers verify and attest to data claims made by *users* [93]. We notice that recent works rely on assumptions (e.g. existence of trustworthy issuers) that go beyond the requirements of SSO systems [55], [86], [94]. Because, in the trust establishment phase of SSO systems, *users* agree to the IDPs's terms and conditions which require *users* to honestly create profiles without requesting specific credentials [21].

Challenge: With this contribution[1] and according to the requirements found in SSO systems, we investigate the honest creation and management of user data, which does not require any form of third-party attestation. In this scenario, we entirely rely on the interaction between *users* and smart contracts, where smart contracts verify the data claims made by *users*. To create a claim on a data sample, *users* convince smart contracts that the data sample complies

[1]Major parts of this Section 3.2 are subject to copyright protection: © 2024 IEEE. Reprinted, with permission, from [Lauinger, Bezmez, Ernstberger, Steinhorst, Portal: Time-Bound and Replay-Resistant Zero-Knowledge Proofs for Single Sign-On, IEEE International Conference on Blockchain and Cryptocurrency (ICBC), May/2024] (cf. Appendix D).

with a public statement. If the smart contract successfully verifies the claim, then the smart contract accepts a mapping between a data claim and the address of the *user*. If *users* create claims on private data, then the smart contracts validate ZKPs asserting the claim. Based on accepted claims, *users* can authenticate to any third party.

We find that replay attacks are a concern because blockchain logs transparently expose ZKP bytes in transaction payloads. Thus, adversaries are able to replay any claim by re-executing the same contract functionality using previously exposed payloads. This is problematic as adversaries can prove statements without access to the private witness that is necessary for the ZKP computation.

Contribution: We show that replay attacks can be prevented. To do so, we introduce a new transaction sequence that unequivocably binds the proof computation to a specific *user* and time (cf. Section 3.2.2). Instead of using a verifier-chosen nonce that binds a proof presentation to a verification session [55], [95], our transaction sequence relies on the blockchain Proof of Stake (PoS) randomness as the verifier-chosen nonce. Our transaction sequence achieves an efficient cost structure as it does not require additional contracts that prevent replay attacks (e.g. access control smart contracts [54]). Based on this contribution, we propose a novel identity system, called *Portal*, which supports on-chain and off-chain validations of ZKPs on *user* data during *user* authentication (cf. bottom part of Figure 3.2). *Portal* targets the research gap to provision an alternative SSO solution, which (i) prevents user tracking through IDPs, and (ii) extends the concepts of SIWE by integrating recent advances of data sovereignty and privacy (e.g. through blockchains and zkSNARK technology [55], [66]). Beyond our initial challenge, we show how *Portal* supports *attestations* in the form of *credentials* and *delegations*. *Credentials* are signed *claims* of a user that have been attested by a trusted issuing third party. *Delegations* embody claims signed by users and are passed from one user to another. *Attestations* may be revoked by the issuing party. *Portal* supports an on-chain and off-chain management of *claims* and *attestations*, and allows *users* to authenticate in the off-chain or on-chain context. In summary,

- Our new transaction sequence secures on-chain ZKP verifications against replay attacks (cf. Section A.1).
- We propose *Portal*, an alternative SSO solution with enhanced privacy and control (cf. Section 3.2.3).
- We open-source[2] our proof of concept of *Portal* and evaluate operation costs (cf. Section 3.2.4).

Disclaimer: In systems with strong Know Your Customer (KYC) requirements, where users cannot be trusted to responsibly operate claims, we want to highlight that *Portal* should be used with third-party attestations.

[2]`https://github.com/jplaui/portal`

3.2.1. System Model

This section defines the data model, system goals, and roles.

Data Model

Key pairs are the *public* and *private keys* of a PKC system.

Addresses are derived from a *user*'s *public key* and exist as 42-character hexadecimal strings appended with '0x'.

Wallets W generate and maintain *key pairs* and, with that, control the *address* W_{addr} corresponding to the *key pairs*.

Data items are key-value pairs, where the key string is a descriptor of the value instance that expresses the data.

Statements ϕ="*key-op-comp*" are strings that express relations between a value $comp$ and a data item with key=*key*. *Statements* use at least one *key*, one operator op (e.g. $>$,$<$,$\neq$,$\overset{?}{=}$,$\in$, etc.) and one comparison value $comp$ (e.g. threshold).

Claims exist as *public claims* claim^{pub}=$\{d, \phi, \mathrm{t}\}$ and as *private claims* claim^{priv}=$\{d, \phi, L, e_{id}, \mathrm{t}\}$. *Public claims* include the *data item* d, a *statement* ϕ, and a timestamp t. If the *data item* of claim^{pub} is stored externally, then d is set to a location identifier d=L. *Private claims* include a *data item* d, a statement ϕ, a location identifier L, an event identifier e_{id}, and a timestamp t. In claim^{priv}, the value instance of d is data hiding (e.g. a commitment string c as d["age"] : c), and the location identifier points to a circuit storage address as L=L_{p_Π}.

Attestations rely on a signature σ=$\textbf{pk.Sign}(t_{exp}, \text{claim}, E_{pk}, id)$ with expiration time t_{exp}, endorser public key E_{pk}, *claim* name name$^{\text{claim}}$, and attestation identifier id. No expiration time is set with t_{exp}=$null$. With the signature parameterization, we define an *attestation* as a tuple a=$\{\sigma, t_{exp}, E_{pk}, \text{name}^{\text{claim}}, id\}$. In *Portal*, *attestations* comprise *credentials* and *delegations*. *Users* issuing *credentials* are called *issuers*. *Credentials* confirm statements that hold on verified *claims*. *Credentials* can be verified by any *third party* that trusts the *issuer*. *Delegations* are marked with ϕ=$null$ and allow a *user* as the delegate to legitimately present a *claim* of another delegating *user*.

Revocations are tuples rev=$\{W_{addr}, t, s_{\text{status}}, id\}$ with the wallet address W_{addr} that holds the revoked *attestation*, a timestamp t, a status string s_{status}, and the revocation identifier id.

Circuits are tuples p_Π=$\{\mathcal{C}, \phi, ccs_{\mathcal{C}}, w_{pub}, pk, vk, L_{\mathcal{C}}\}$, where the compiled constraint system $ccs_{\mathcal{C}}$ expresses a provable representation of a circuit $\mathcal{C}$ that implements the assertions expressed by the *statement* ϕ. To assert *statements*, the circuit $\mathcal{C}$ evaluates private inputs w to a representation that can be compared against public inputs w_{pub}. The prover and verifier keys pk,vk are created by running the setup algorithm **zk.Setup** of a proof system Π. If the verification call of the circuit $\mathcal{C}$ is deployed as a *smart contract*, then the locator $L_{\mathcal{C}}$ is set to the *address* of the *circuit contract*. Otherwise, $L_{\mathcal{C}}$=$null$.

Transactions are tuples tx=$\{\sigma, d_{pl}, t_{addr}, g_{used}\}$ with a signature σ from the transaction sender, a data payload d_{pl}, a *gas* value g_{used} and a destination *address* t_{addr}. *Transactions* are used to invoke and pay for *smart contract* calls at an address t_{addr} and provide non-repudiation of the transaction sender.

Registry contracts C^{reg} maintain a map $m[W_{addr}]C^{id}_{addr}$ linking registered *wallet addresses* and *addresses* of *identity contracts*. C^{reg} exposes a *register* method which requires the transaction payload to include an *identity service* signature on a new *identity contract address*. Further, for the identification of circuits, C^{reg} maintains a map $m[\text{name}^{\mathcal{C}}]L_{p_\Pi}$ which associates location identifiers of circuit parameters L_{p_Π} with circuit names $\text{name}^{\mathcal{C}}$.

Identity contracts C^{id} maintain the parameters *claims*, *attestations*, and *revocations* in the respective maps $m[\text{name}^{\text{claim}}]\text{claim}$, $m[\text{name}^{\text{att}}]a$, and $m[a_{id}]\text{rev}$, where a_{id} is an attestation identifier. Claim and attestation names $\text{name}^{\text{claim}}, \text{name}^{\text{att}}$ are unique strings.

Circuit contracts $C^{\mathcal{C}}$ verify ZKPs on-chain and emit events e_{id} according to the outcome of a ZKP verification. *Circuit contracts* expose the *sample* and *verify* methods. If a *transaction* calls the *sample* method, then $C^{\mathcal{C}}$ associates a PoS randomness as a nonce with the wallet address of the *user* in a map $m[W_{addr}]nonce$. The randomness is used during the *verify* method which verifies a ZKP.

System Roles

Users hold *wallets*, deploy *identity contracts*, and register the *address* of the *identity contract* at the *registry contract* after passing an authenticity verification at the *identity service*. *Users* individually manage *claims* and *attestations*, and authenticate themselves at *third-party services* by linking or presenting data. *Users* count as *issuers* in the context of signing and sharing credentials with other *users*.

Identity services deploy and maintain *registry* and *circuit contracts* and connect *users* to the *Portal* identity system. We envision non-profit organizations to take the role of the *identity service* and assume that *identity services* have the expertise to create secure ZKP circuits that evaluate *claims* of *users*.

Third-party services (e.g. web services) authenticate *users* based on the *Portal* identity system and trust *issuers*.

Blockchain networks provide decentralized and verifiable computation and storage through *smart contracts* and manage *registry*, *identity*, and *circuit contracts*.

Storage networks provide decentralized, fault-tolerant, and high-availability storage of data at locations L and are used to store larger data objects such as circuit parameters p_Π.

System Goals

Sybil resistance prevents an adversary from registering an arbitrary amount of pseudonymous identities.

Decentralized resolution guarantees that the storage and computation of user data remain publicly verifiable, trustless, and available towards a resolving *third-party service*.

```
$\mathbf{assertClaim}(d,\ \mathrm{p}^{\mathrm{MT}},\ \mathrm{W}_{addr},\ \mathrm{n};\ \mathbf{root}^{\mathrm{MT}},\ \mathbf{W}_{addr},\ \mathbf{n},\ \phi)$:
1. assert: $\mathbf{n} \overset{?}{=} n;\quad W_{addr} \overset{?}{=} \mathbf{W}_{addr};\quad 1 \overset{?}{=} f_{\phi}(d)$
2. return: $1 \overset{?}{=} \mathbf{cs.Open}(\mathrm{p}^{\mathrm{MT}},\ \mathrm{d},\ \mathbf{root}^{\mathrm{MT}})$
```

Figure 3.3.: ZKP circuit to verify a *data item d* of a private claim against a MT commitment $\mathrm{root}^{\mathrm{MT}}$. The MT has a depth of 5 and a path p^{MT} as the private witness. The circuit has 9.29K constraints and evaluates d against ϕ="d[age]->-18" using the function f_{ϕ}. The semicolon **;** separates private inputs (left of **;**) from boldly formatted public inputs (right of **;**). © 2024 IEEE

On-chain & off-chain verification of private data allows *users* to (i) present data to a *third-party service*, where the data has been verified at smart contracts or (ii) interactively convince a *third-party service* of a data verification. The on-chain management of data guarantees that the storage and computation of user data remain publicly verifiable, trustless, and available.

User-centricity gives *users* full control to manage data. Transactions in *Portal* are independent of *identity services* or *third-party services* and depend on *users* and *issuers* only.

Accountability gives attesting parties the option to revoke an *attestation*. The *user* remains publicly accountable with respect to revoked *attestations*.

Data-provenance allows *users* to present *claims* to *third-party services*, such that the origin and correctness of the presented *data item* can be publicly verified.

Cost-efficiency optimizes operation costs for *third-party* and *identity services* and enables scalability of *Portal* with cheap maintenance costs for the *identity service*.

Threat Model

We assume that transactions sent to blockchain nodes are secured via TLS such that the TLS properties of message confidentiality, integrity, and authenticity hold. We assume that (i) honest *users* are able to resolve the correct state s_t of the blockchain at time t, that (ii) collision-resistant hash functions are used in the blockchain PoS protocol to determine the block randomness [96], and (iii) active, adaptive, and PPT adversaries that are able to perform machine-in-the-middle (MITM) attacks and intercept communication traffic. Adversaries are not able to block traffic indefinitely and cannot modify intercepted traffic. Adversaries have access to the *mempool*, can access transaction payloads by observing blockchain logs, and replay transactions tx or ZKPs of a *user*.

3.2.2. Constructing Time-bound and Replay-resistant ZKPs

In the following paragraphs, we outline why ZKP verification at smart contracts is insecure. Subsequently, we propose a solution to the highlighted problem.

ZKP Verification at Smart Contracts

As the initial setup, we assume access to a circuit tuple p_Π, which has been instantiated by a trusted party p_0. The party p_0 derives the solidity verification code of $\mathcal{C}_1 \in p_\Pi$ for the creation and deployment of a circuit contract $\mathcal{C}^{\mathcal{C}_1}$ (cf. steps 1.4, 1.5 of Figure 3.4). Π uses a ZKP system which compiles the circuit $\mathcal{C}_1$. The circuit $\mathcal{C}_1$ performs an address and nonce check, asserts a private *data item* against a statement ϕ, and checks if the *data item* computes to a public commitment string (cf. **assertClaim** logic of Figure 3.3). Now, a *user* as party p_1 is able to compile transactions with a payload that contains the bytes of a ZKP π, and call the deployed contract $\mathcal{C}^{\mathcal{C}_1}$ for an on-chain verification of π.

Binding ZKP Computations to the PoS Randomness

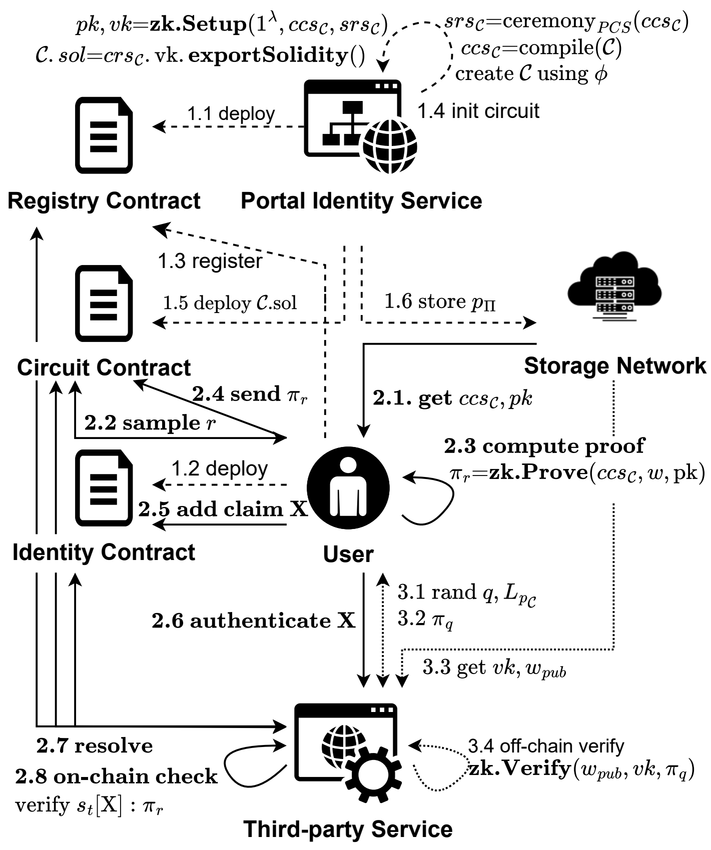
In the following, we define a transaction sequence where a user p_1 compiles the transaction tx_1 to call the *sample* method of the contract $\mathcal{C}^{\mathcal{C}_1}$. Upon receiving tx_1, $\mathcal{C}^{\mathcal{C}_1}$ associates the latest PoS randomness r with the *user*'s wallet address by depositing both parameters into the map $m[W_{addr}]nonce$. Initially, the randomness is concatenated with a state string to represent the nonce as $nonce{=}s_t.prevrandao||$"-0". After $\mathcal{C}^{\mathcal{C}_1}$ samples the nonce, *users* fetch and use the deposited nonce to compute a ZKP π using the circuit $\mathcal{C}_1$. To prevent replay attacks and ensure time-bound proofs (cf. Section A.1), the ZKP circuit $\mathcal{C}_1$ takes in and compares both the *user*'s wallet address and the nonce as private inputs and public inputs. Notice that binding values (e.g. the nonce) to a ZKP computation via public inputs is secure [95]. In a transaction tx_2, p_1 calls the *verify* method of $\mathcal{C}^{\mathcal{C}_1}$, which upon a successful verification of π, sets the nonce to $m[W_{addr}]s_t.prevrandao||$"-1" and emits an event with an identifier e_{id}. If party p_1 presents e_{id} towards any *third-party service*, then the *third-party service* can use e_{id} to resolve and verify a successful on-chain ZKP verification via smart contract logs (cf. steps 2.1-2.8 in Figure 3.4). We show the security of our construction in Appendix A.1.

3.2.3. *Portal* Identity System

In the following subsections, we introduce the *Portal* protocol specification to manage *claims* and *attestations*, and show how *sybil resistance* and *data-provenance* is achieved.

Architectural Considerations

To achieve a scalable system design, all operation costs are shifted towards the *user*. The reason to shift operation costs to the *user* has two benefits. First, the cost for the *identity service* is optimized to a minimum of paying the transactions to deploy the *registry* and *circuit contract*. Secondly, as all transactions of the *Portal* system are issued by *users*, *users* remain responsible and accountable for every action in the system. Even though any party is able to pre-process, deploy, and store circuit parameters p_Π, we see *identity services*, as *Portal* system maintainers, as suitable providers of p_Π, and rely on the assumption that *identity services* have the most expertise to construct secure ZKP circuits. Finally, depending on the size of managed *data items* and *circuits*, we outsource storage to an external *storage network* (e.g. IPFS [34]).



pk, vk=**zk.Setup**$(1^\lambda, ccs_\mathcal{C}, srs_\mathcal{C})$
$\mathcal{C}.sol$=$crs_\mathcal{C}$. vk. **exportSolidity**()
$srs_\mathcal{C}$=ceremony$_{PCS}(ccs_\mathcal{C})$
$ccs_\mathcal{C}$=compile($\mathcal{C}$)
create $\mathcal{C}$ using ϕ
1.1 deploy
1.4 init circuit
Registry Contract
Portal Identity Service
1.3 register
1.5 deploy $\mathcal{C}$.sol
1.6 store p_Π
Circuit Contract
Storage Network
2.4 send π_r
2.1. get $ccs_\mathcal{C}, pk$
2.2 sample r
2.3 compute proof
π_r=**zk.Prove**$(ccs_\mathcal{C}, w, \text{pk})$
1.2 deploy
2.5 add claim X
Identity Contract
User
3.1 rand $q, L_{p_\mathcal{C}}$
3.2 π_q
2.6 authenticate X
3.3 get vk, w_{pub}
2.7 resolve
2.8 on-chain check
verify s_t[X] : π_r
3.4 off-chain verify
zk.Verify(w_{pub}, vk, π_q)
Third-party Service

Figure 3.4.: *Portal* architecture to manage a *private claim*. The system deployment, *user* registration, and the circuit pre-processing is indicated with dashed arrows (1.1-1.6). The on-chain verification of *private claims* at time t_1, and *private claim* presentation towards a *third-party service* is depicted with solid lines (2.1-2.8). The live verification at time $t > t_1$ of a *private claim* is indicated with dotted lines (3.1-3.4). To add a *public claim*, the *user* does not interact with the circuit contract. © 2024 IEEE

Bootstrapping

To bootstrap the *Portal* identity system, the *identity service* deploys a *registry contract* C^{reg}, which sets the *identity service* as the owner of the contract. With that, the *identity service* withholds the rights to invalidate the contract C^{reg} or remove registered *users*. Further, the *identity service* pre-processes ZKP circuits $\mathcal{C}$ and stores circuit parameters p_Π at location identifiers L_{p_Π} (e.g. a IPFS Uniform Resource Locator (URL)) of the *storage network*. If the circuit $\mathcal{C}$ verifies ZKPs on-chain, then the *identity service* deploys a *circuit contract* $C^\mathcal{C}$ to the public blockchain before storing p_Π externally. Notice that *users* or *issuers* are equally eligible to deploy and maintain circuit parameters. However, we deem the *identity service* as the most suitable party to manage ZKP circuits, and we anticipate that the *identity service* has the expertise to compile secure ZKP circuits.

Registration

The registration of a new *user* in the *Portal* system depends on two transactions. The first transaction deploys the *identity contract* C^{id} of the *user*. In the same way as the *registry contract*, the constructor of the *identity contract* sets the deploying party as the owner of the contract. Only the owner of C^{id} is able to call methods that modify the state of C^{id}. In the *Portal* system, *users* and *issuers* are both represented by *identity contracts*, where *users* modify *claims* and *attestations*, and *issuers* modify *revocations*.

The compilation of the second transaction requires the *user* to obtain a signature $\sigma_{C^{id}_{addr}}$ from the *identity service* on the *identity contract address*. Before signing any C^{id}_{addr}, the *identity service* verifies and deduplicates *users*, such that *sybil resistance* holds. *Users* use the second transaction to invoke the *register* method at the *registry contract* C^{reg}, which checks the signature validity of $\sigma_{C^{id}_{addr}}$ before including the *user*'s wallet address and C^{id}_{addr} into the map $m[W_{addr}]C^{id}_{addr}$. If the *user* shares the wallet address W_{addr} with any *third-party service*, then the *third-party service* is able to resolve C^{id}_{addr} via the map of the registry contract. From here, the *third-party service* is able to obtain *claims* and *attestations* of the *user*. Before resolving any data via a *user*'s wallet address, the *third-party service* verifies W_{addr} address ownership through a signature challenge. In the same way as the SIWE sign-in challenge [27], our signature challenge demands the *user* to compute a signature on a randomly sampled nonce with the corresponding wallet *key pair*, where the nonce is sampled by the *third-party service*.

Claim Management

1. **Public Claims** claim^{pub} are created by *users* and added to the *identity contract* C^{id} by calling the claim^{pub} method. If *public claims* are stored on-chain, then resolution times are fast. Due to increasing transaction costs of claim^{pub} for larger data sizes, our system supports external storage of claims at locations L_s (e.g. IPFS URL) by setting $d{=}L_s$. If data is stored externally, resolution times depend on the network delay of the *storage network*.

2. **Private Claims** claim^{priv} can be managed in two distinct ways:

   On-chain Verification In the first mode, which we call *on-chain* mode, *users* send two transactions to the *circuit contract* C^{C} (cf. solid lines 5,7 in Figure 3.4). The first transaction calls the *sample* method, where C^{C} associates the latest PoS randomness with the *user*'s wallet address by depositing both parameters into the map $m[W_{addr}]nonce$. Initially, the randomness is concatenated with a state string to represent the nonce as $nonce{=}s_t.prevrandao||''\text{-}0''$. After C^{C} samples the nonce, *users* fetch and use the deposited nonce to compute a ZKP. As already discussed in Section 3.2.2, this transaction sequence prevents replay attacks and ensures time-bound proofs.

   Notice: Replaying publicly accessible data cannot be prevented which is why this work solves replay attacks of claims made on private data. However, if an adversary operates claims on public data and cheats (uploads false claims), then the blockchain properties guarantee that adversaries remain accountable once misbehavior is detected. In this case, the reputation of a *user* declines.

Once $C^{\mathcal{C}}$ successfully verifies a ZKP at time t, $C^{\mathcal{C}}$ sets the nonce at address W_{addr} to $m[W_{addr}]s_t.prevrandao||$"-1" and emits an event with an identifier e_{id}. Using e_{id}, *users* compile and add the *private claim* claim^{priv} to the *identity contract* C^{id}. With that, any *third-party service* is able to verify *private claims* via *smart contract* logs.

Live Verification The on-chain mode has one issue. If *private claims* are verified by *smart contracts*, then *claims* can be outdated in the eyes of the *third-party service*. As a remedy, the *Portal* system has the *live* mode which supports verification of ZKPs at verifier-chosen times. For that, the *Portal* plugin (cf. Section 3.2.3) at the *third-party service* generates and shares a fresh nonce with the *user* at time t_1 and selects a location identifier of circuit parameters L_{p_Π}. Before the circuit parameters are shared, the *third-party service* resolves all *Portal* supported circuits through the map $m[\text{name}^{\mathcal{C}}]L_{p_\Pi}$ at C^{reg}. With L_{p_Π}, *users* fetch circuit parameters from the *storage network* and compute a ZKP with proof π^{t_2} at time t_2, where $t_2 > t_1$. The verifier-chosen nonce binds the ZKP computation to time-bound session [55] (cf. dotted lines in Figure 3.4).

Attestation Management

Similar to how we classify claims, we consider attestations of two different types:

1. **Credential**: *Attestations* as *credentials* provide *issuers* with the option to attest to *claims* of *users*. *Issuers* share attestations with *users* after verifying the *data item d* of a claim against the statement ϕ of the claim. When attesting *private claims*, *issuers* are supposed to verify statement compliance through a ZKP circuit which is indicated via the circuit location identifier L_{p_Π} of the *claim*. Once the *user* obtains the attestation a, the *user* is able to add the attestation to the *identity contract* C^{id} by calling the *setAtt* method. By sending the *setAtt* transaction to C^{id}, the *user* agrees to the attestation of the *issuer*. To check the validity of the signature σ_{claim} through the *Portal* plugin, *issuers* must create an *identity contract* C^{id} and sign *claims* with the corresponding *key pair*.

2. **Delegation**: An *attestation* as a *delegation* provides a *user* u_1 with the option to delegate the rights of presenting a *claim* owned by u_1 to other *users* u_x. *Users* u_x agree to and add delegations a to the *identity contract* by calling the *setAtt* method. To check delegations via the *Portal* plugin at the *third-party service*, both the delegating *user* u_1 and *users* u_x as delegates require *identity contracts*.

Revocation: The *issuer* or delegating *user* withholds the rights to revoke any issued *attestation*. To compile a revocation rev, the attesting party sets the status string to s_{status}="revoked" and sets the wallet address of the attested *user u* as W^{u}_{addr}. The moment at which the revocation is effective is indicated by the revocation timestamp t. Every revocation is added to the *identity contract* of the *issuer* or delegating *user* by calling the *setRev* method.

Portal Plugin

The *Portal user* authentication is handled by a single software plugin that runs at *third-party services*. The software plugin serves the seamless integration of the *Portal* identity system into the traditional web infrastructure and, for the *third-party service*, reduces the *Portal* points of contact to a single module. The software plugin comprises client libraries that connect to and resolve data from the *storage network* and the blockchain *smart contracts*. Further, we equip the plugin with ZKP back-end systems for the *live* verification of ZKPs.

An *attestation* is verified by the *Portal* plugin at the *third-party service* through a sequence of actions. If any verification action fails, then the plugin aborts. If a *user* presents a wallet address W_{addr} and an attestation name name^{att}, then the plugin takes the known and fixed address of the *Portal registry contract* C^{reg} to resolve the *user*'s *identity contract* through the map $m[W_{addr}]C^{id}_{addr}$. With name^{att}, the plugin identifies the attestation entry a in C^{id} and obtains the signature, the expiration time, and the claim name. After verifying the expiration time, the plugin uses the claim name to resolve the claim, and the endorser *public key* E_{pk} to resolve the *identity contract* of the attesting party. Notice that the wallet address of the endorser W_{addr} can be derived from the endorser public key E_{pk}. With access to the C^{id} of the attesting party, the plugin verifies revocation entries which are identified by the claim identifier a_{id}. If the revocation checks pass and no revocation entry is found, then the plugin continues to verify the attested signature of the resolved claim by using the public key E_{pk}.

Data Provenance

The *Portal* system allows seamless integration of data provenance services (e.g. TLS oracles). Data provenance ensures enhanced data correctness and allows the *third-party service* to verify if the presented *Portal* credential originated from a specific website and complies with an oracle-specific statement (e.g. ϕ="$d[\text{balance}_{\text{paypal}}]>5k$") [11], [12], [58], [97]. To obtain a data provenance *credential*, *users* interact in a verification challenge defined by the TLS oracle protocol, where the oracle verifier acts as the credential *issuer* in the *Portal* system. After the *user* adds the credential as an *attestation* and the *third-party service* trusts the oracle verifier, then data provenance can be provided for private and public claims.

3.2.4. Evaluation

The evaluation uses a MacBook Pro with the Apple M1 Pro chip and 32 GB of Random Access Memory (RAM). The benchmarks average ten executions of the same experiment.

Implementation

The *Portal* proof of concept was conducted locally using the *Ganache*[3] test network (*v7.8.0*) as the public blockchain. To manage a decentralized external storage network, we rely on the containerized *Kubo*[4] IPFS implementation. Due to our local deployment with a single IPFS

[3]https://github.com/trufflesuite/ganache
[4]https://github.com/ipfs/kubo

Table 3.1.: *Portal* Deployment and Transaction Costs. © 2024 IEEE

Tx	Type	Cost (eth/$)	Time (ms)	Size (kB)
C^{reg}	deploy	4.15e-3/8.65	18	bc:6.5,tx:6.6
C^{id}	deploy	6.5e-3/13.56	10	bc:10.2,tx:10.7
$C^{\mathcal{C}_1}$	deploy	4.96e-3/10.29	**385**	bc:7.4,tx:12.4
set_$\mathcal{C}_1$	call C^{reg}	8.4e-5/0.18	11	tx: 0.46
register	call C^{reg}	7.4e-5/0.16	51	tx: 0.3
claim^{pub}	call C^{id}	6.4e-05/0.13	3	tx: 0.48
sample	call $C^{\mathcal{C}_1}$	6.6e-05/0.14	6	tx: 0.1
verify_π	call $C^{\mathcal{C}_1}$	8.4e-4/**1.76**	**252**	tx: 1.20
claim^{priv}	call C^{id}	3.9e-4/0.82	21	tx: 0.68
setAtt	call C^{id}	4.01e-4/0.84	27	tx: 0.39
setRev	call C^{id}	3.45e-4/0.72	9	tx: 0.36

node, we disabled InterPlanetary Name System (IPNS) to achieve faster resolution delays when searching for stored data via content hashes. We rely on the solidity compiler *solc v0.8.20* as the PoS block randomness *prevrandao* is available in all versions above *v0.8.18*. We develop a *Portal* Golang System Development Kit (SDK) to deploy and maintain *Portal* at every party and use the official Ethereum repository *go-ethereum*[5] including *abigen v1.10.16* to interact with smart contracts. We convert transaction costs into dollars based on the rate 2084.42$ per 1 *eth* (Nov. 2023) and select the gas price gas_{price}=28gwei according to the gas price of the Ethereum network[6]. We select the Golang *gnark* (*v0.9.1*) repository [98] as the ZKP system and configured (i) the *plonk* backend with a universal setup to verify ZKPs on-chain, and (ii) the *plonkFRI* backend with a transparent setup for the live verification of ZKPs. To prove and store private *claims* efficiently, we benchmark the ZKP circuit $\mathcal{C}_1$ (cf. Figure 3.3), which evaluates *data items* of claims^{priv} as private input against a MT commitment as the public input. We use the MiMC hash function [99] to compress the MT data. We open-source the *Portal* code with the *smart contracts* and simulation scenarios in the repository[7].

Costs Analysis

Table 3.1 shows the *Portal* cost analysis, where transaction costs compute as $\text{tx}_{\text{cost}}=gas_{\text{used}} \cdot gas_{\text{price}}$. Depending on the transaction type, Table 3.1 measures byte code and transaction sizes. We explain the execution times in the range of milliseconds with the local deployment of *Portal*. By deploying *Portal* on the Sepolia[8] testnet, we measured transaction resolution times taking around 150ms and transaction calls taking between 1.3s ($C^{\mathcal{C}_1}$ deploy) and 9.4s (setNonce+verify_π+claim^{priv}). We explain higher execution times of the transactions that

[5]https://github.com/ethereum/go-ethereum
[6]https://etherscan.io/gastracker#chart_gasprice
[7]https://github.com/anonsubsub/portal
[8]https://www.alchemy.com/overviews/sepolia-testnet

deploy $\mathcal{C}_1$ and verify a proof of $\mathcal{C}_1$ with the corresponding higher transaction sizes. Compared to other contracts, which initialize empty maps, the byte code of $C^{\mathcal{C}_1}$ stores large cryptographic parameters which increase the transaction size of $C^{\mathcal{C}_1}$. Except for transactions of the type deployment and the transaction to verify a ZKP on-chain, the cost per transaction remains below 1$. Thus, as contracts are deployed once, we consider *Portal* as cost-efficient.

Simulation Scenarios

We present additional microbenchmarks of *Portal* in the top part of Table 3.2 and show that *Portal* supports the storage and resolution of data via the blockchain (row 3) or an external storage network (row 1). For larger data sizes, such as circuit parameters (row 2), *Portal* must rely on IPFS to remain practical and cost-efficient. Lower data sizes can be stored and resolved efficiently via the blockchain (cf. rows 3-5). The lower part of Table 3.2 presents end-to-end benchmarks per system role for two simulation scenarios: the (i) on-chain and (ii) live mode of a private claim. The numbers of the simulation scenarios build upon microbenchmarks provided in the top part of Table 3.2 and involve transaction benchmarks of Table 3.1.

For the *identity service* (IDS), setting up *Portal* via $\text{setup}_{\text{e2e}}$ depends on the tuple of transactions (C^{ref},$C^{\mathcal{C}_1}$,set_$\mathcal{C}_1$), the **zk.Setup** call, and to storage call setIPFS^{p_Π}. Notice that to instantiate *Portal*, $\text{setup}_{\text{e2e}}$ must be called once. Registering a *user* via $\text{register}_{\text{e2e}}$ requires the transactions C^{id} and register. Setting a *private claim* via $\text{setClaim}^{priv}_{\text{e2e}}$ depends on calling the tuple of transactions (setNonce, verify_π, claim^{priv}), resolving the tuple (nonce=res^{nonce}, p_Π=getIPFS^{p_Π}), and calling **zk.Prove**. Finally, with $\text{showClaim}^{priv}_{\text{e2e}}$, the *user* presents a private claim to the *third-party service* (TPS) with the tuple (W_{addr}, $\text{name}^{\text{claim}}$) in a TLS session that takes 20ms. The on-chain verification of a private claim concerns the tuples of resolutions ($\text{res}^{W_{\text{addr}}}$, $\text{res}^{\text{claim}}$) and would depend on ($\text{res}^{\text{attest}}$, res^{rev}) when verifying an *attestation*. The live verification of a private claim requires the *user* and TPS to download p_Π and locally call **zk.Prove** and **zk.Verify** with *plonkFRI* respectively. Before the live verification, the *third-party service* discovers available ZKP circuits by resolving $\text{res}^{\mathcal{C}_1}$.

The simulation results show that the on-chain verification of private claims trades a one-time transaction cost at the *user* for cheap presentation and verification computations. In contrast, the live verification introduces reoccurring calls of **zk.Prove** and **zk.Verify** and a communication cost in the size of megabytes. However, when relying entirely on live verification, *users* are less susceptible to tracking. The evaluation of the ZKP circuit $\mathcal{C}_1$ using the *plonkFRI* backend yields a 3.1s setup, 3.4s prove, and 1ms verification time. Even though *plonkFRI* increases execution times, *Portal* relies on *plonkFRI* for any ZKP live verification to leverage the transparent setup benefits that come with *plonkFRI*.

3.2.5. Discussion

This section compares *Portal* against related works (cf. Table 3.3) and summarizes how future work addresses current limitations.

Table 3.2.: *Portal* Microbenchmarks and Simulation Scenarios. © 2024 IEEE

Name	Roles	Time (ms)	Com (B)
set / getIPFS$^{\text{claim}}$	*user*/TPS-IPFS	73 / 4	pub:136,priv:210
set / getIPFS$^{p_\Pi}$	IDS-IPFS	**631 / 66**	**7.43 (MB)**
res$^{\text{claim}}$	TPS-BC	**5**	**pub:136,priv:210**
res$^{W_{\text{addr}}\ /\ \text{nonce}}$	*user*/TPS-BC	10 / 6.2	42 / 78
res$^{\text{attest / rev / }\mathcal{C}_1}$	TPS-BC	4.7/4.5/4.8	120 / 56 / 130
zk.Setup$^{\mathcal{C}_1}_{plonk}$	IDS	1029	$p_\Pi^{\mathcal{C}_1}$=7.43 (MB)
zk.Prove$^{\mathcal{C}_1}_{plonk}$	*user*	195	π=552
zk.Verify$^{\mathcal{C}_1}_{plonk}$	TPS	2.5	-

Name	Cost ($)	Time (ms)	Com (B)
setup$_{\text{e2e}}$	19.12	2074	7449.4 (kB)
register$_{\text{e2e}}$	13.72	61	11 (kB)
setClaim$^{priv}_{\text{e2e}}$	2.72	546.2	7432.6 (kB)
showClaim$^{priv}_{\text{e2e}}$	-	**20**	**50**
verifyClaim$^{priv}_{\text{e2e}}$	-	15	252
showClaim$^{live}_{\text{e2e}}$	-	**3507**	**7430.5 (kB)**
verifyClaim$^{live}_{\text{e2e}}$	-	71.8	7430.1 (kB)

Related Works

The work DecentID [100] introduces a *smart contract* identity system that resolves user data via four different contracts. DecentID supports access control of user data via a key management protocol and discloses user data entirely upon access provision. In *Portal*, *users* can control and provide access privileges by creating and sharing multiple *identity contracts*. In contrast to DecentID, *Portal* supports enhanced data privacy through on-chain and off-chain ZKP computations.

The work *zk-creds* [55] proposes the first anonymous zkSNARK credentials with an unlinkable multi-show presentation, where linkage gadgets obfuscate links between subsequent proofs of the same credential. With a verifier-chosen nonce, *zk-creds* prevents credential replays towards the verifier in the off-chain context. By contrast, *Portal* works on-chain and applies the PoS randomness to prove zkSNARK claims in unique and replay-resistant verification sessions.

The work Zebra [54] introduces a zkSNARK credential scheme with an on-chain ZKP verification at an access control contract. Before a *user* authenticates at an application smart contract with a wallet address W_{addr}, the *user* posts a ZKP to the access control contract to provide access privileges to W_{addr}. In contrast to Zebra, *Portal* does not entirely focus on credentials and introduces the concept of user-accountable claims. With claims, *Portal* provides a solution that matches SSO requirements. Further *Portal* improves the cost-efficiency by

Table 3.3.: Comparison of *Portal* with Related Works. © 2024 IEEE

Paper	Dec. Resolution	On/Off-chain Verify	Extra Contract
DecID	✓	✗ / ✗	✓
zk-creds	✗	✗ / ✓	✗
Zebra	✓	✓ / ✗	✓
zkLogin	✗	✓ / ✓	✗
Portal	✓	✓ / ✓	✗

solving ZKP replay attacks via a cheap transaction sequence, instead of relying on additional smart contracts.

The work *zkLogin* [101] modifies the *OpenID Connect* nonce by computing a public key on the random nonce value from the third-party IdP (e.g. Google). The resulting value is used to derive an on-chain address which the *user* can prove using a ZKP. Proving the generated address allows the *user* to authenticate transactions in the on-chain context. Thus, the *user* is able to leverage legacy identity providers and use existing *OpenID Connect* credentials in smart contracts. Portal tries to minimize the reliance on third-party entities (e.g. *OpenID Connect* providers) and does not focus on the generation of session tokens (e.g. *OpenID Connect* nonce values).

Limitations & Future Work

Portal runs on the native blockchain network called Layer 1 (L1). To optimize transaction costs, we envision deploying *Portal* via scalable Layer 2 (L2) networks (e.g. zk-rollups [102]). We expect that our proof-of-concept implementation requires minor adjustments to reach L2 compatibility as existing tooling for L2 deployments exists. With a L2 deployment, *Portal* must be compared towards related works with regard to cost and efficiency. Another item of future work is the security assessment under relaxed assumptions of blockchain properties and the consideration of censorship implications. Concerning decentralizing the *identity service*, we either (i) register *users* based on a multi-party signature issued by multiple *identity services*, or (ii) maintain a list of public keys in the *registry contract*, such that public keys authorize *identity services*. We like to highlight that *Portal* is compatible with data provenance solutions if *users* interact with attesting oracle services [58], [97]. To align *Portal* with standardization efforts, we see *OpenID Connect*, W3C DID, and VC as appealing compliance goals.

3.3. Privacy-preserving Authorization for Sovereign Identity

Sovereign credential management promotes self-determined control of credentials without relying on external parties who administer data. However, existing solutions of sovereign credentials (e.g. DIDs and VCs as defined by the W3C) do not enable credential holders to verify whether a Credential Issuing Authority (CIA) legitimately issued a credential. As a remedy, we construct a secure authentication protocol, called A-PoA, to provide privacy-preserving authorization of CIAs. We leverage a cryptographic accumulator to enable Root Authorities (RAs) with the ability to authorize CIAs to issue credentials. CIAs can prove accumulator membership via a non-interactive zero-knowledge proof. This allows a credential holder or blockchain validator to verify the validity of a CIA, while the CIA remains anonymous. Our security analysis shows the integrity and confidentiality of our protocol against malicious adversaries and our experimental evaluation shows constant verification complexity independent of the number of authenticated CIAs that are registered in the accumulator. Hence, with A-PoA, we introduce the missing building block to develop certification chains in sovereign credential architectures that are compatible with the Verifiable Credential (VC) ecosystem.

Recap & Motivation: The term identity management (IdM) refers to data management around the identification, authorization, and authentication of identifiers in any form. In recent years, different forms of IdM have emerged. Central IdM system design maintains the identity of users in a single system and continues to remain vulnerable to the single point of failure pattern [22]. Federated IdM systems distribute data of identities across trusted platforms and provide benefits of exchange and linking of identities. However, such systems enforce trade-offs between transparency, and usability, and negatively affect user privacy [24].

More recent solutions target user-centric IdM to keep users or devices in full control of their identity data, removing the dependence on third parties. Self-Sovereign Identity Management (SSIM), as a promising approach to user-centric IdM, enforces user-controlled attributes to bring a new notion of independent data administration. In combination with a VC, the SSIM scheme can protect private information with enhanced trust [28].

In the context of the VC ecosystem (cf. Section 2.1.2), a VC relies on a:

- **Credential Schema (CS)**, which specifies the name, version, and attributes that will appear in the credential.
- **Credential Definition (CD)**, which references a CS and specifies cryptographic metadata containing the signatures and data of the CIA.

The cryptographic data in a CD will be used for verifying attributes of an issued credential. Hence, a CD enables a third party to cryptographically verify the validity of claims made in a credential by the respective CIA. Therefore, to issue a credential, a CIA must release a CD beforehand (cf. Figure 3.5).

Challenge: As RAs and CIAs have equal privileges to write to the verifiable registry (decentralized public infrastructure), the credential holder remains unable to verify whether a

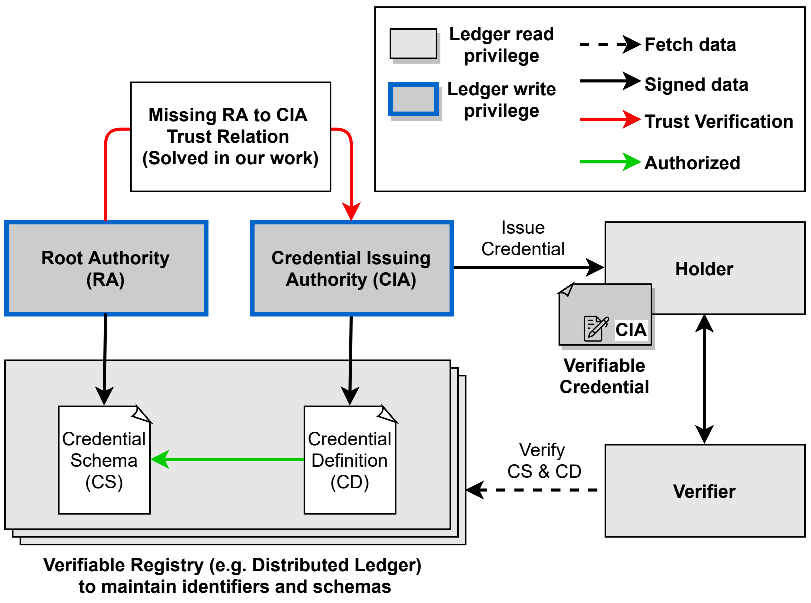


Figure 3.5.: Presentation of the ecosystem around Verifiable Credentials (VCs), highlighting the missing relation between the Root Authority (RA) (Schema Creator) and the Credential Issuing Authority (CIA) (Definition Creator & Credential Issuer). Our work specifies a protocol for the establishment of a trust relation between RAs and CIAs. © 2021 IEEE

CIA is authorized to issue a credential. This issue remains an unsolved problem in the VC standardization by the W3C and sets the core motivation for our contribution.

To clarify the problem with a use case in the automotive domain (cf. Figure 3.5), assume a vehicle Original Equipment Manufacturer OEM_i as a RA (CS issuer). The CS, published by OEM_i, would specify attributes of a credential that is autonomously verified during a software update by On-Board Equipment (OBE). Without an authorization scheme for CIAs, every Software Provider SP_j would be able to register CDs that reference the CS of OEM_i. Thus, every SP_j would be able to issue credentials to Software Distribution Services SDS_k^j (acting as credential holders). A vehicle (acting as verifier), manufactured by OEM_i, would autonomously verify certificates of every SDS_k^j as valid. Even if the update is malicious and is connected to a maliciously acting SP_j.

Contribution: To solve this issue, we propose an authorization scheme, called Anonymous Proof of Authorization (A-PoA), which enables RAs (such as OEM_i) to authorize CIAs (such as contracted SP_j) to issue credentials based on CSs that have been created by RAs. The main feature of our protocol is that A-PoA keeps the relation between RAs and CIAs anonymous to entities that verify the privilege authenticity of CIAs. To securely construct this protocol, our A-PoA protocol makes use of the membership verification feature of the cryptographic RSA-accumulator, introduced in the work [40]. The accumulator allows to aggregate elements without revealing individual membership. Additionally, we apply the Non-interactive Zero

Knowledge Proof of Knowledge of Exponent (NI-ZKPoKE) construction of the work [45] to hide the accumulator elements in the membership verification phase of our protocol. By disguising the membership authentication with the NI-ZKPoKE, A-PoA does not reveal any structure of accumulator elements, hence, keeping the anonymity of authenticated CIAs.

The evaluation of our work analyzes the security of our protocol and we benchmark the performance. Concerning the security of our protocol, integrity holds throughout all phases of the protocol whereas confidentiality partly applies. The performance evaluation shows efficient benchmarks for the registration, authentication, and revocation phases. With the requirement of fast and scalable verification, A-PoA achieves constant verification times which we assume to happen more frequently compared to registration or revocation operations.

Summed up, the contribution of our work[9] is as follows:

- In Section 3.3.2, we construct our protocol that enables RAs (CS creators) to authorize and revoke CIAs (CD creators) for the issuance of verifiable credentials.
- Our A-PoA protocol efficiently verifies CIA privileges by leveraging dedicated ZKPs (cf. Section 3.3.2).
- Our security analysis evaluates the protocol integrity, confidentiality of specific accumulator parameters, and the anonymity of authorized CIAs (cf. Section A.2).
- We enable *verifiable* and *anonymous* trust hierarchies of issuing parties for sovereign VC ecosystems.

3.3.1. Extended System Model

The following system model extends the system model defined in Section 3.2.1. This involves an extension of the threat model, the introduction of additional system roles, and different system goals.

Notations

We define all necessary notations in the Table 3.4, which introduces roles, accumulator parameters, and cryptographic parameters.

System Roles

Root Authorities are additional parties that are supposed to create credential schemas (CS).

Credential Issuing Authorities are supposed to create credential definitions and take the role of issuing users in the *Portal* system model. Further, we assume that the CIAs issue credentials according to publicly available CDs.

[9]Major parts of this Section 3.3 are subject to copyright protection: © 2024 IEEE. Reprinted, with permission, from [Lauinger, Ernstberger, Regnath, Hamad, Steinhorst, A-PoA: Anonymous Proof of Authorization for Decentralized Identity Management, IEEE International Conference on Blockchain and Cryptocurrency (ICBC), May/2021] (cf. Appendix D).

Table 3.4.: Glossary of Notations: (1) Roles, (2) Accumulator Parameters, and (3) Arithmetic Modulo Primes & Composites. © 2021 IEEE

Symbol	Definition
RA_i	Witness issuing (i) Authorities (CS creator)
CIA_h	Witness holder (h) Authorities (CD creator)
H_i	Credential Holders
VN_i	Validator Nodes / Verifiers
CS_i	Credential Schemas
CD_i	Credential Definitions
t	Discrete time / operation counter
X_t	Tails file at time t
X_0	Initial tails file at $t = 0$
x_i	i-th element of the tails file
w_i	Witness value associated with i-th element
a_t	Accumulator value at time t
p, q, p', q'	Large λ-bit prime numbers
$\mathbb{Z}_n$	$n \in \mathbb{N}$, $\mathbb{Z}_n = \{1, 2, \ldots, n\}$ = ring of integers mod n
$\mathbb{Z}_n^*$	Set of invertible elements in $\mathbb{Z}_n$
$\mathbb{G}_?$	Generic group of unknown order $\{(\mathbb{Z}_n)^*/\{\pm 1\}\}$
$[-B, B]$	Range of integers such that $\lvert\mathbb{G}\rvert/B$ is negligible
$\mathbb{QR}_n$	Subgroup of quadratic residues of $\mathbb{G}_?$, contains $x \in \mathbb{Z}_n^*$, if $\exists\, y \in \mathbb{Z}_n^*$, with $y^2 \equiv x \pmod{n}$
$\phi(n)$	Number of elements in $\mathbb{Z}_n^*$, if $p \cdot q = n$ then $\phi(n) = (p-1)\cdot(q-1)$
g, h	Generator of a Group $\mathbb{G}_?$

Accumulator Managers (Smart Contract) are supposed to publicly manage the cryptographic accumulator and we consider a smart contract at a decentralized public infrastructure as the accumulator manager.

Credential Holders are supposed to request, obtain, and present credentials. Credential holders take the role of *Portal* users who manage claims.

Credential Verifiers take the role of third-party services in the *Portal* system model and verify incoming credentials of credential holders.

System Goals

Anonymity holds for the identities of CIA parties that are registered in the cryptographic accumulator. This means that no adversary is capable of linking issuing parties during the verification of credentials.

Integrity holds such that no malicious adversary can obtain a false authorization privilege from the RA.

Threat Model

Throughout the security analysis, we assume to have the following models of adversaries:

- $\mathcal{A}_1$ *(Network Eavesdropper)*: Suppose a hostile network participant, acting as $\mathcal{A}_1$, intends to eavesdrop and modify or decrypt all messages m exchanged throughout the introduced protocols.
- $\mathcal{A}_2$ *(Unforgeability)*: Suppose $\mathcal{A}_2$ is a malicious adversary trying to forge a valid proof of an invalid identity. $\mathcal{A}_2$'s efforts can be based on previously seen witness pairs (x, w) (only w is known by $\mathcal{A}_2$) and accumulator values a.
- $\mathcal{A}_3$ *(Cheating Verifier)*: Suppose $\mathcal{A}_3$ is a malicious Verifier V that verifies the authentication proofs of a prover P. Then, $\mathcal{A}_3$ does not learn anything else than the validity of the statement proven by P.

Further, we assume a safe accumulator manager that does not share the tails file over the network. Additionally, no hostile network participant is able to forge the private keys associated with DIDs in use. Network communication is secured using authenticated encryption and session keys are random to prevent replay attacks.

3.3.2. A-PoA Protocol Specification

This section describes the A-PoA protocol that solves the missing trust relation between trusted authorities in the VC ecosystem. Section 3.3.2 provides an overview of A-PoA and introduces different phases around the accumulator management. The subsequent sections go into the details of each protocol phase.

Protocol Overview

In A-PoA, we associate an element x with a CIA and the accumulator value a_t with a specific CS (cf. Section 2.2.2 for details on the accumulator construction). Since RAs create CSs, RAs count as accumulator managers. By adding an element x to the accumulator value a_t at time t, RA authorizes a CIA to issue a credential based on the specific CS. During the CIA registration phase, the CIA, as the creator of the element x, receives a witness w. A witness w can be extracted from a_t for every element x in a_t. With that, it is possible to verify the existence of x in a_t by using the respective witness w_t for x. The element-witness pair (x, w) enables the CIA to prove membership of x in a_t. Thus, authenticating itself as an authorized authority to issue a credential based on a specific CS, which has been created by the accumulator manager RA. The RA maintains a so-called tails file to monitor authenticated CIAs in the form of elements x. The fact that it is possible to remove an element x from the accumulator a_t provides RAs with the ability to revoke authenticated CIAs at any point in time.

The RSA-accumulator we use has been introduced in Section 2.2.2. Necessary functions to manage the accumulator can be seen with the first five functions of Figure 3.6. The last two functions of Figure 3.6 generate and verify the NI-ZKPoKE proof of an element-witness pair (x, w). All functions apply at different stages throughout A-PoA and will be explained in the next sections. To enable anonymous access control of CSs in the VC ecosystem, we divide A-PoA into four main phases:

1. **Schema Registration** requires a RA to publish a CS as well as an associated accumulator to the verifiable registry.
2. **CIA Registration and Authorization** requires the accumulator manager RA to add an element to the accumulator and return a witness.
3. **CIA Authentication** requires the CIA (witness holder) to generate an authentication proof for the CD transaction.
4. **Maintenance** describes incoming addition and revocation updates of the accumulator which affects the authorization status of CIAs.

Schema Registration

Before we introduce our approach to authorize CIAs, it is necessary to set up the initial state of the verifiable registry via a smart contract. Each node maintaining the smart contract may have several pre-defined transactions defining the initial pool of network nodes. It is assumed that authorities $\{\mathrm{RA}_1, \mathrm{RA}_2, ..., \mathrm{RA}_i\}$ are initialized at the smart contract with writing permission. With that, each public entity can be initialized through a special transaction on the verifiable registry which discloses their public DID. It is assumed that with each initialized CS_i, created by RA_i, an accumulator a_t is generated and registered at an accumulator registry A_t on the verifiable registry (cf. *GenAcc()* in Figure 3.6). Likewise, RA_i creates an empty or initialized tails file X_0 at time $t = 0$. The schema registration protocol can be seen in the first phase of Figure 3.7 which stops at the first dotted horizontal line.

CIA Registration and Authorization

As the first core part of our work, this section describes the secure generation of an accumulator element x_h and its corresponding witness w_h. The interacting authorities are the witness issuer RA_i (CS_i creator & accumulator manager) and the witness holder CIA_h (CD_i creator & credential issuer). Both witness issuer and holder have the privilege to write to the verifiable registry. We assume that the state of the verifiable registry is initialized as described above. To achieve collision resistance among accumulator elements, we utilize a hash function H_{prime} with prime domain. This hash function iteratively hashes an input and a counter to generate λ-bit accumulator elements. Increasing the counter eventually creates an output of the hash function that is prime and co-prime to $\phi(n)$ [45]. Note that this function can be the target of timing and side-channel attacks. Thus, H_{prime} leverages using dummy computations to prevent such attacks.

```
1. GenAcc($\lambda, X_0$):
2. $p' \leftarrow Gen_{\text{prime}}(\lambda)$;    $q' \leftarrow Gen_{\text{prime}}(\lambda)$
3. $p \leftarrow 2 \cdot p' + 1$;    $q \leftarrow 2 \cdot q' + 1$
4. $g'_{\text{acc}} \overset{R}{\leftarrow} \mathbb{G}_?$;    $n \leftarrow p \cdot q$
5. $g_{\text{acc}} = (g'_{\text{acc}})^2 \bmod n$;    $a_t = g_{\text{acc}}^{\prod_{i=0;x\in X_0}^{N} x_i} \bmod n$
6. return: $a_t$
7. GenAccElement($\lambda$):
8. $x \overset{R}{\leftarrow} \mathbb{Z}$
9. return: $H_{\text{prime}}(x, \lambda)$
10.Add($a_t, X_t, x_i$):
11. if $x_i \in X_t$: return: $(X_t, a_t)$
12. else:
13. $X_{t+1} \leftarrow X_t \cup \{x_i\}$
14. $a_{t+1} = a_t^{x_i} \bmod n$
15. return: $(X_{t+1}, a_{t+1})$
16.Revoke($a_t, X_t, x_i$):
17. if $x_i \notin X_t$: return: $(X_t, a_t)$
18. else:
19. $a_{t+1} = a_t^{x_i^{-1} \bmod \phi(n)} \bmod n$
20. $X_{t+1} \leftarrow X_t \setminus \{x_i\}$
21. return: $(X_{t+1}, a_{t+1})$
22.GenWit($x_i, X_t, g_{\text{acc}}$):
23. $w_t = g_{\text{acc}}^{\prod(X_t \setminus \{x_i\})} \bmod n$
24. return: $w_t$
25.UpdateWit($w_t, x_i, rev, a_t, x_k^{\text{deleted}}$):
26. if $rev == true$:
27. $\alpha \cdot x_i + \beta \cdot x_k^{\text{deleted}} = 1$; ($\alpha$,$\beta$ Bezout)
28. $w_{t+1} = w_t^{\beta} \cdot a_t^{\alpha}$
29. return: $w_{t+1}$
30. else:
31. $w_{t+1} = w_t^{x_i} \bmod n$
32. return: $w_{t+1}$
33.GenProof($w_x, x, a_t$):
34. $k, \rho_x, \rho_k \overset{R}{\leftarrow} [-B, B]$;    $z = g^x h^{\rho_x}$
35. $A_g = g^k h^{\rho_k}$;    $A_{w_x} = w_x^k$
36. $l \leftarrow H_{\text{prime}}(w_x, a_t, z, A_g, A_{w_x})$;    $c \leftarrow H(l)$
37. $q_x \leftarrow \lfloor (k + c \cdot x)/l \rfloor$;
$q_\rho \leftarrow \lfloor (\rho_k + c \cdot \rho_x)/l \rfloor$
38. $r_x \leftarrow (k + c \cdot x)$
 $\bmod l$; $r_\rho \leftarrow (\rho_k + c \cdot \rho_x) \bmod l$
39. $\pi \leftarrow \{l, z, g^{q_x} h^{q_\rho}, w_x^{q_x}, r_x, r_\rho\}$
40. return: $\pi$
41.VerifyProof($w_x, a_t, \pi$):
42. $\{l, z, Q_g, Q_{w_x}, r_x, r_\rho\} \leftarrow \pi$;    $c = H(l)$
43. $A_g \leftarrow Q_g^l g^{r_x} h^{r_\rho} z^{-c}$;    $A_w \leftarrow Q_{w_x}^l w_x^{r_x} a_t^{-c}$
44. Verify $r_x, r_\rho \in [l]$;
$l = H_{\text{prime}}(w_x, a_t, z, A_g, A_w)$
45. return: $\{0, 1\}$
```

Figure 3.6.: Pseudocode of the RSA-accumulator generation (*GenAcc*), element generation (*GenAccElement*), addition (*Add*), subtraction (*Revoke*) [39], and witness functions (*GenWit* & *UpdateWit*) [40] together with the verification based on the NI-ZKPoKE protocol (*GenProof* & *VerifyProof*) [45]. © 2021 IEEE

The protocol for authorizing a CIA_h to issue credentials based on CS$_i$ (cf. intermediate phase in Figure 3.7) is as follows:

- **GenAccElement:** CIA$_h$ chooses an element $x'_h \in \mathbb{Z}$ at random and calculates a λ-bit $x_h \leftarrow H_{\text{prime}}(x'_h, \lambda)$ (see Figure 3.6, line 9). CIA$_h$ sends x_h to RA$_i$.
- **Add:** RA$_i$ adds x_h to the local tails file X_t and updates the accumulator $(X_{t+1}, a_{t+1}) \leftarrow Add(a_t, X_t, x_h)$. RA$_i$ updates the corresponding accumulator value a_t and writes the updated accumulator value a_{t+1} to the verifiable registry.
- **GenWit:** RA$_i$ calculates a new witness $w_h \leftarrow GenWit(x_h, X_{t+1}, g_{\text{acc}})$ and sends w_h to CIA$_h$.

As the element x_h will be written into the tails file X_t (granting CIA$_h$ access to reference

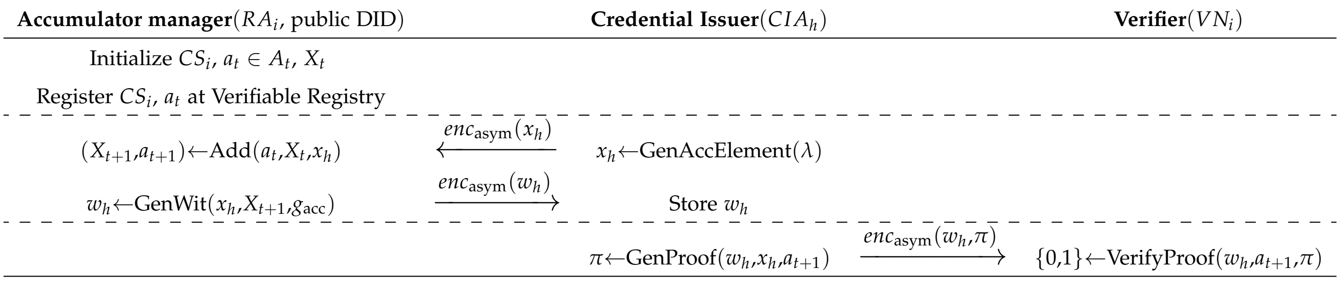

Accumulator manager$(RA_i$, public DID$)$
Credential Issuer(CIA_h)
Verifier(VN_i)
Initialize CS_i, $a_t \in A_t$, X_t
Register CS_i, a_t at Verifiable Registry
$(X_{t+1}, a_{t+1}) \leftarrow \text{Add}(a_t, X_t, x_h)$
$enc_{\text{asym}}(x_h)$
$x_h \leftarrow \text{GenAccElement}(\lambda)$
$w_h \leftarrow \text{GenWit}(x_h, X_{t+1}, g_{\text{acc}})$
$enc_{\text{asym}}(w_h)$
Store w_h
$\pi \leftarrow \text{GenProof}(w_h, x_h, a_{t+1})$
$enc_{\text{asym}}(w_h, \pi)$
$\{0,1\} \leftarrow \text{VerifyProof}(w_h, a_{t+1}, \pi)$

Figure 3.7.: A-PoA protocol of membership authorization and verification. © 2021 IEEE

CS$_i$) which is saved in the wallet of RA$_i$, x_h values need to be a pairwise distinct prime values to ensure collision-resistance as described in the work [43]. The tails file X_t of RA$_i$ contains a mapping of the DID of CIA$_h$ to the value aggregated in the accumulator a_t.

As RA$_i$ saves X_t in its own private wallet, witnesses will need to be recomputed every time a value is added or deleted from the accumulator. This is why accumulator updates introduce communication overhead as RA$_i$ needs to multicast updated witnesses back to each CIA$_h$. As our focus lies on efficient verification where authorities are assumed to have sufficient computational power in the network, computational overhead is negligible.

CIA Authentication

As the second core part of this contribution, this section introduces the authentication of CIA$_h$. Successful authentication results in the authorization of CIA$_h$ to perform CD$_i$ transactions, enabling credential issuing. The third phase of Figure 3.7 illustrates this phase of our protocol with the interaction between CIA$_h$ and VN$_i$, acting as prover and verifier respectively.

When intending to issue credentials, CIA$_h$ creates a CD$_i$ transaction with a valid NI-ZKPoKE proof of knowledge of x_h. The function *GenProof()* of Figure 3.6 shows the proof construction which has been developed in [45]. Including the NI-ZKPoKE proof which proves membership of x_h in a_t, a CD$_i$ transaction is sent to the verifiable registry. Here, a validator node VN$_i$ verifies if CIA$_h$ is entitled to issue credentials referencing a CS$_i$ of RA$_i$. This verification is achieved by verifying the NI-ZKPoKE proof at a smart contract. The NI-ZKPoKE proof does not disclose the actual value x_h and preserves the confidentiality of the parameter x_h. Hence, there is no loss of privacy for CIA$_h$. Line 34 of Figure 3.6 indicates how the Pedersen commitment hides x_h. Again, non-interactivity is achieved through leveraging the Fiat-Shamir heuristic [103] (cf. Appendix C), which models the unpredictability of the random choices of VN$_i$ through the output of a hash function. Thus, the protocol for proving membership of x_h in a_t is the following:

- **GenProof:** CIA$_h$ generates the proof $\pi \leftarrow \mathit{GenProof}(w_h, x_h, a_t)$, where $x_h \in X_t$, and sends (w_h, π) to the verifier VN$_i$.
- **VerifyProof:** VN$_i$ verifies the membership by invoking $\{0, 1\} \leftarrow \mathit{VerifyProof}(w_h, a_t, \pi)$.

Once VN$_i$ verifies the membership of CIA$_h$, CD$_i$ is written to the verifiable registry, effectively enabling CIA$_h$ to issue credentials based on a CS$_i$ issued by RA$_i$. An important remark is that the authentication of CIA$_h$ should be based on the NI-ZKPoKE proof only. Likewise, the privilege of a CD$_i$ write transaction should rely on a valid NI-ZKPoKE proof. This means that CIA$_h$ must take a random DID for the communication with VN$_i$ instead of using the publicly known and trusted DID.

Maintenance

Updating the witness of an authority has to be done each time a member is added or revoked from the accumulator tails file X_t. The witnesses can only be updated by RA$_i$. Therefore, the accumulator manager/RA$_i$ sends an update message to all the members that are a part of the updated accumulator a_{t+1}. We distinguish two cases for the witness update protocol:

Addition: *UpdateWit(rev=false)* updates a witness w_t when a member is added to X_t. Therefore, *UpdateWit* adds the new element to the witness w_t, which itself is an accumulator value with an element less compared to the actual accumulator a_t. Addition of elements $\{x_{i+1}, x_{i+2}, ..., x_j\}$ (members of the accumulator) to the tails file $X_t = \{x_1, x_2, ..., x_i\}$ requires execution of $X_{t+1} = X_t \cup \{x_{i+1}, x_{i+2}, ..., x_j\}$. This can be seen in line 13 of Figure 3.6. With X_t, the witness owner can calculate each witness by calculating $w_t = g_{\text{acc}}^{\prod(X_{t+1} \setminus \{x_i\})}$ individually. After the updates, each CIA$_h$ receives its updated witness value.

Revocation: With the accumulator scheme, authority RA$_i$ is able to revoke the trust from CIA$_h$ by invoking $(X_{t+1}, a_{t+1}) \leftarrow \mathit{Revoke}(a_t, X_t, x_h)$. This removes the element x_h associated with CIA$_h$ from the tails file X_t. Next, a new accumulator value is computed and written to the verifiable registry, effectively revoking the ability for CIA$_h$ to prove its accumulator membership to the verifier VN$_i$ (smart contract) or the credential holder. Additionally, CIA$_h$ loses its ability to further issue credentials. Note that all credentials, which have already been issued during the time where CIA$_h$ was authorized, are only invalidated if the credential contains the proof that allows credential holders to verify the validity (CS authorization) of RA$_i$.

Efficiently updating the membership witness upon deletion of x_k^{deleted} can be achieved by calling *UpdateWit(rev=true)*. This function call removes x_k^{deleted} from the accumulator by calculating the Bezout coefficients between x_i and x_k^{deleted} as described in [104]. The Bezout coefficients always exist since the domain $\mathbb{Z}_N^*$ of the RSA-accumulator contains odd prime integers only. Thus, the updated membership witness w_{t+1} can be computed such that the Bezout coefficients α and β solve the linear equation $\alpha \cdot x_i + \beta \cdot x_k^{\text{deleted}} = 1$ for $gcd(x_i, x_k^{\text{deleted}}) = 1$. Lines 27 and 28 of Figure 3.6 show the calculation of the Bezout coefficients and the witness update. A complete description and correctness proof of the preceding relation is provided in [104].

3.3.3. Evaluation

This evaluation considers three aspects of our A-PoA construction: (1) aggregated protocol times, (2) scalability, and (3) privacy. After introducing hardware specifications and selected bit-sizes, we evaluate the protocol performance and scalability.

Table 3.5.: Mean execution times (ms) of A-PoA with a 2048-Bit RSA- accumulator ($\lambda = 128$), k=50 elements, and 128-Bit hashes. © 2021 IEEE

Protocol	Function	Time (ms)	COM	Big $\mathcal{O}$
Authorization	Prime Gen.	309.81	k	$\mathcal{O}(n)$
	Acc. Gen.	88.80	1	$\mathcal{O}(1)$
	Wit. Gen.	4383.60	k	$\mathcal{O}(n)$
Authentication	GenProof	40.06	1	$\mathcal{O}(1)$
	VerifyProof	23.42	0	
Revocation	Acc. Revoke	2244.85	(0-k)	$\mathcal{O}(n)$
$\sum_{\textbf{Authorization}}$	N/A	4782.21	$2k+1$	$\mathcal{O}(n)$
$\sum_{\textbf{Authentication}}$	N/A	63.48	1	$\mathcal{O}(1)$
$\sum_{\textbf{Revocation}}$	N/A	2244.85	(0-(k-1))	$\mathcal{O}(n)$

Timing Behavior of the RSA-Accumulator and Protocols

The performance evaluation has been conducted using a Lenovo Thinkpad T480s with 16 GB of RAM and a 1.90 GHz Intel(R) Core(TM) i7-8650U CPU and we used Python to implement A-PoA. All values collected during the evaluation of the accumulator data structure and protocols faced 100 repetitions and subsequent averaging to reduce deviations in the results.

Concerning the selection of parameters, this contribution takes accumulators with an RSA modulus n with 1024-bit and 2048-bit sizes, requiring $\lambda = 80$-Bit and $\lambda = 128$-Bit primes respectively. The sizes of the parameters of the modulus n and primes do not only affect the accumulator functions but the NI-ZKPoKE protocol and its proof size, calculating as $\pi_{\text{size}} = 3 \cdot n_{\text{size}-\text{bit}} + 3 \cdot \text{Hash}_{\text{size}-\text{bit}}$. This means that taking the generic group of unknown order $\mathbb{G}_? = \mathbb{Z}_n^* \setminus \{\pm 1\}$ with a 2048-Bit modulus n and a 128-Bit hash-function (H_{prime}, H), the NI-ZKPoKE proof size π_{size} calculates to 861 Bytes for the 2048-Bit RSA-accumulator. The formula of π_{size} derives from the Zero-Knowledge (ZK) proof with parameters $\{l, z, Q_g, Q_{w_x}, r_x, r_\rho\}$, where z, Q_g, and Q_{w_x} depend on the modulo n calculation (2048-Bit n), l depends on the output size of H_{prime} which causes r_x and r_ρ (remainders) to remain below l. Switching the modulo size of the RSA-accumulator and the hash output size affects π_{size} accordingly.

Protocol Times - Table 3.5 shows execution times of functions of the authorization, authentication, and revocation protocols for managing access to a CS. Additionally, the last column provides the communication overhead presented by each function. The evaluation of protocol times is based on an accumulator with $k = 50$ elements x_i. This decision enables comparison to the mean computation times of the protocols (revocation, verification) introduced by the work [105] which we further discuss in the upcoming related works paragraph. Concerning the communication overhead, if 50 CIAs participate in the protocol requires 50 messages of

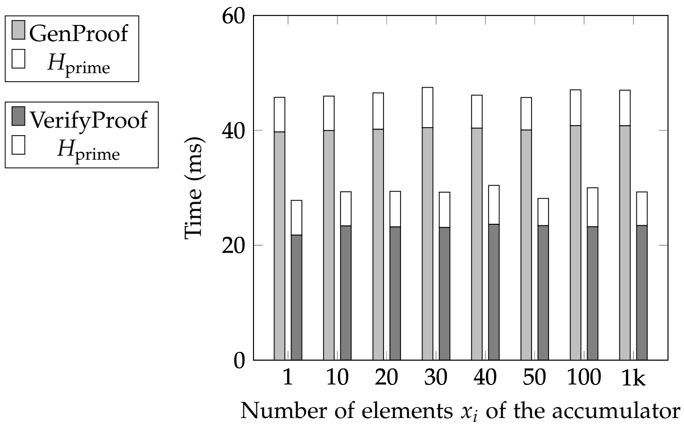


Figure 3.8.: GenProof (left, lightgray) and VerifyProof (right, gray) execution times (ms) of the NI-ZKPoKE protocol with 128-Bit polynomial time H_{prime} hash function and the RSA-accumulator (2048-Bit). © 2021 IEEE

each CIA$_h$ to the accumulator manager RA$_i$, 1 message of RA$_i$ to the verifiable registry, and again, 50 messages to communicate each witness w_h back to every CIA$_h$. The communication complexity of the revocation protocol depends on the application and the decision whether to notify CIAs. Due to the non-interactivity feature of the NI-ZKPoKE protocol, the authenticity verification is efficient with a single message.

Proof Verification - An evaluation of different numbers of elements aggregated in the accumulator with respect to the corresponding verification and proof creation time can be found in Figure 3.8. On average and without considering H_{prime} (volatility caused by non-deterministic behavior), the proof verification is 42% faster than the proof creation for the 2048-bit RSA modulus n with a hash size of 128-bit and a security value $\lambda = 128$. We assume that the accumulator value has already been fetched from the verifiable registry and the verification is restricted to evaluating the *VerifyProof()* algorithm introduced in Figure 3.6. The values depicted in Figure 3.8 are averaged over 100 executions. Notably, the verification speed is constant ($O(1)$) for constant system parameters and does not depend on the number of elements aggregated in the accumulator. The times of executing H_{prime} vary and the optimization of this function goes beyond the scope of this work.

Witness Update - In our construction, the accumulator manager RA authorizes CIAs to access CSs through broadcasting witness updates back to authorized CIAs. The efficiency of the witness update operation depends on the number of elements that are added to or deleted from the accumulator at a given point in time. The addition of 10 elements to the witness of an authorized entity takes around 20 ms with a 2048-bit RSA modulus n and around 3.5 ms by using a 1024-bit RSA modulus n (cf. Figure 3.9). Addition and deletion of CIAs do not occur very often as CIAs itself maintain their issued certificates at least the expiry time (90 days validity of Let's Encrypt certificates) [106]. Nevertheless, the communication overhead in A-PoA is a bottleneck that could be solved by having CIAs update their witnesses on their own or by having them rely on third parties. With our anonymity requirement, outsourcing of

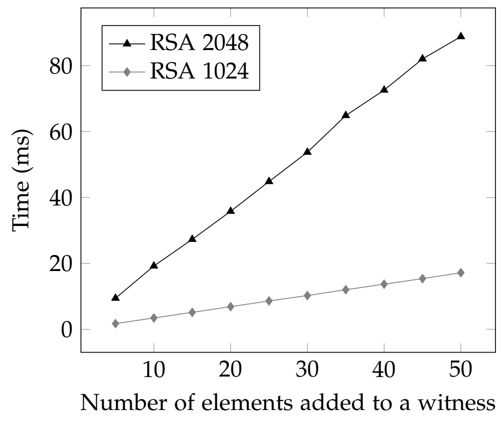


Figure 3.9.: Duration (ms) of adding elements x_i to an existing witness w_t for a single holder witness update (numbers averaged by 100 repetitions). © 2021 IEEE

protocol computations is not possible, and the bottleneck remains. However, leveraging more complex accumulators such as the *Braavos* accumulator of the work in [39] do not require updates upon addition of elements (but has weaker security guarantees), and we consider the usage of such accumulators as future work.

Discussion

Privacy: Resulting of specifications of A-PoA, no entity except the accumulator manager/RA and the authenticated CIA can reveal or track information of the CIA witness pair (x, w). The reason for this is the confidential communication between the RA and CIA authorities as well as the accumulator data structure itself. The aggregated elements in the RSA-accumulator are secure under the discrete logarithm problem and, by respecting the strong RSA assumption, face negligible collision chances. When proving membership of a CIA to an accumulator, the NI-ZKPoKE proof hides the accumulator element without revealing any structure of it. It is important to notice that the CIA must constantly switch its DIDs, even when communicating to the same entity periodically to prevent information tracking. This way, the authorized/privileged CIA remains anonymous.

Related Work: The work of Hölzl et al. [107] leverages a so-called disposable dynamic accumulator in the context of a pseudonym-based signature scheme where pseudonyms are represented by tokens. The accumulator data structure of their work proves the validity of these tokens that make up the accumulator elements. In a similar way to our validity management of authorities per CS, the Electronic Identity (*eID*) issuer in [107] creates a single accumulator per secure element which handles credential storage. One-time verification tokens as pseudonyms and accumulator elements establish the privacy-preserving functionality, whereas in our work, accumulator computations and ZK-proofs provide anonymity.

The evaluation of the work of Hölzl et al. considers generation, binding, verification, and update times of a RSA-accumulator in the context of mobile *eID* management. Similar to their work, our authorization and revocation methods increase/decrease linearly depending on the number of elements in use. Figure 3.9 shows this behavior for a changing number of elements x_i. By contrast, our execution times of the NI-ZKPoKE proof generation and verification remain constant (see Figure 3.8). Our NI-ZKPoKE verification times are up to 25% faster compared to the verification times in the work [107].

In [108], Reyzin et al. utilize asynchronous accumulators with backward compatibility to build a distributed PKI. In their work, the accumulator is used to reference and store public keys of users. The witness value and the public key allow validity checking of security key pairs. However, compared to our work and with regard to a PKI, our concept is able to leverage hierarchies and anonymity proofs to manage user credentials with absolute privacy and user-centric control of credentials.

4. Policy-driven Data Provenance

In this chapter, we investigate the limits of privacy-enhancing technologies (PETs) in the context of verifying the provenance of data. Data provenance protocols are an important building block to build data sovereign systems and allow devices to reclaim control and authority over data that resides at external online services. In our online world of today, most digital data lies in online services, due to the fact that the Internet has used external identity services to manage data. Moving the control and ownership of online data back to the devices is not as easy as it seems. If data on the Internet could be witnessed transparently, then assessing the provenance of digital data is easy and does not require specific solutions. However, most countries enforce the use of secure communication protocols for the exchange of online data. And, secure communication channels prevent external parties from witnessing exchanged data. Thus, if a device requests data from a centralized data hub and presents that data to another third party, then the third party cannot verify the origin of the data. As a solution to this problem, data provenance protocols enable devices to present protected data to third parties such that the third party is able to verify the provenance of the data. As such, devices regain the power to export digital assets and reputation to any new service of choice.

Our contributions [58], [97] solve important limitations of current data provenance protocols. Before we outline our contributions, Section 4.1 systematizes the state of the art of existing data provenance solutions. The initial analysis clarifies our direction of research and motivates why we focus on improving software-based data provenance protocols. Next, our first contribution of this chapter (cf. Section 4.2) analyzes the main efficiency bottlenecks that affect online execution times of data provenance protocols. Based on our insights, we propose two new building blocks which (i) enhance the computation times of ZKPs at end devices and (ii) reduce the overheads introduced by secure two-party computation techniques during the TLS record phase. Our first contribution facilitates the deployment of secure data provenance protocols in constrained environments. Our second contribution of this chapter (cf. Section 4.3) targets the reduction of bandwidth requirements and enables an Internet-wide rollout of data provenance validators. Our second contribution reconsiders the security assumption made by existing data provenance approaches. By relying on a weaker network adversary, we propose a protocol that does not depend on any multi-party computation (MPC) techniques. Instead, we show that an extended ZKP circuit counteracts emerging security risks that occur in a data provenance setting without MPC. As a third contribution [109] (cf. Section 4.4), we develop a policy-driven generation of secure computation circuits (ZKP circuits and MPC circuits), which automates a dynamically configured, policy-compliant verification of data provenance.

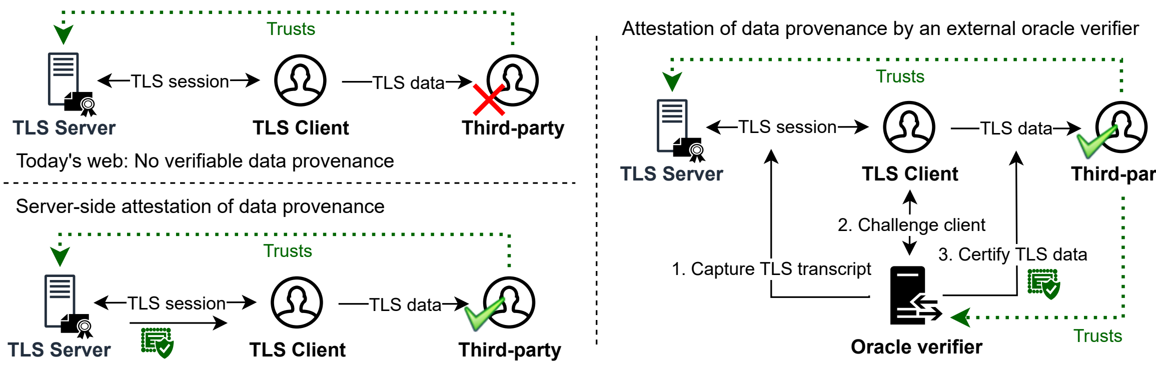


Figure 4.1.: High-level overview of data provenance solutions. The illustration of TLS sessions in today's web is depicted in the top left corner. TLS sessions, per default, are secure channels between two parties and prevent a third-party from verifying the provenance of TLS data. Data provenance protocols solve this issue and certify TLS data such that TLS data becomes publicly verifiable. TLS data is either attested on the server side (cf. bottom left corner) or by an additional external party (cf. right side).

4.1. Systematization of TLS Oracles

In this section, we give a brief systematization of existing approaches for verifying the provenance of data. In the scope of this thesis, we focus on the provenance of TLS data because TLS is the most used secure communication channel on the Internet. Notice that our contributions equally apply to other secure channel protocols (e.g. Secure Shell (SSH) protocol). Data provenance protocols that operate on TLS as a secure channel are referred to as TLS oracles. In the following, we interchangeably use the names TLS oracles or data provenance protocols and refer to the same thing.

As initially pointed out (cf. TLS in Section 2.2.3), TLS establishes a confidential and authenticated communication channel between two parties. As a consequence, without the opportunity to witness exchanged data in the secure channel, no extra third party is able to verify whether presented data originated from the secure channel (cf. top left corner in Figure 4.1). Third parties are only able to verify the provenance of TLS data if a trustworthy party attests to the correctness of shared TLS data. Naturally, to prevent self-signed attestations on claims, the attesting party cannot collude with the party presenting the TLS data. As a result, two options remain for the attestation of TLS data. Either the server attests to data that has been exchanged during the TLS session (cf. bottom left context of Figure 4.1), or another extra party takes over the validation of TLS data integrity according to a valid TLS session (cf. right side of Figure 4.1). If the extra party challenges the client using a ZKP system, then data privacy can be maximized such that the oracle verifier learns nothing beyond a statement validity on the TLS data. In the following, we discuss security, sovereignty, and usability properties that exist in different settings for verifying data provenance.

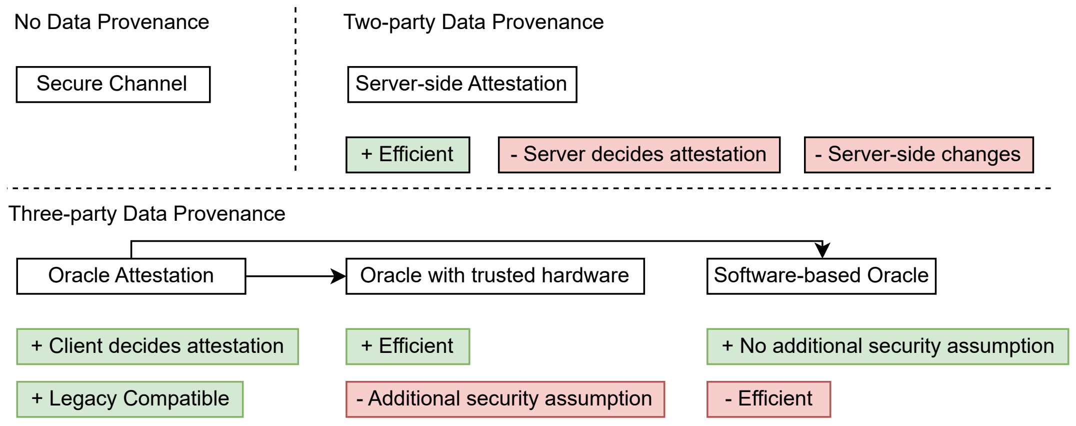

No Data Provenance
Secure Channel
Two-party Data Provenance
Server-side Attestation
+ Efficient
- Server decides attestation
- Server-side changes
Three-party Data Provenance
Oracle Attestation
Oracle with trusted hardware
Software-based Oracle
+ Client decides attestation
+ Legacy Compatible
+ Efficient
- Additional security assumption
+ No additional security assumption
- Efficient

Figure 4.2.: Systematization of TLS oracles with regard to different deployment settings, security assumptions, and usability properties.

4.1.1. System Settings, Assumptions, Properties

The features of TLS oracles depend on the number of involved parties, the access to security assumptions, and the deployment options. The next paragraphs iterate over all existing configurations and highlight consequential effects.

Two-party Setting (Server-side Attestation)

In the two-party setting, the TLS server attests to the contents exchanged during the TLS session [50], [110]. This approach is highly efficient because the signature generation on the TLS contents is highly efficient. A major disadvantage of the server-side attestation is that the attestation mechanism imposes software changes on the server. Rolling out individual software changes at TLS endpoints costs resources and can lead to misconfigurations and resulting vulnerabilities. Concerning data sovereignty for TLS clients, the two-party setting is weak as the system provides the server with the control over deciding which data to attest. And, in many cases (e.g. attesting to digital art or to external values), web endpoints manage device data where the correctness cannot be verified. For data that is managed and verified entirely at the server (e.g. payment history and transactions), servers can attest to data states. The powers of data attestation shifts in the three-party setting of TLS oracles which we discuss next.

Three-party Setting (External Verifier)

In the three-party setting, the attestation of TLS data is handled by an additional external party, called the Oracle verifier. Here, the verifier challenges the client with a TLS data presentation which eliminates potential security attacks that are performed by the client. From the data sovereignty perspective, this challenge is important as it empowers devices to decide the scope of data attestation. Further, the three-party setting counts as *legacy compatible* as it does not introduce server-side changes. Instead, attestations performed by an external oracle verifier operate with the functionalities provided by TLS.

Three-party TLS oracles can be constructed using trusted hardware enclaves [111], which provide secure computation with input privacy. Even though trusted hardware computing is highly efficient, multiple attacks on trusted hardware environments exist [112], labeling the trusted hardware security assumption as problematic. Hence, it is of interest to remove the additional security assumption of trusted hardware and, instead, rely on security assumptions achieved by software implementing cryptography. As such, software-based TLS oracles [12] do not require additional security assumptions beyond hardness guarantees of software security (cf. Section 2.2.1). Unfortunately, building efficient software-based TLS oracles remains difficult as today's PET building blocks drastically increase network overheads and slow down execution times.

Deployment Options

With respect to the deployment mode in the three-party setting, the verifier can either act as an external third-party behind the *client* (*notary mode*) or appear in between the client and the server as a proxy (*proxy mode*). If the Oracle verifier takes the role of a proxy, then servers can detect and consecutively block proxies that are used to withdraw credible data. On the other hand, the proxy mode facilitates the deployment of Oracle verifiers because browsers allow the configuration of proxies.

Blocking the functionality of TLS oracles is not possible if the Oracle verifier runs as a notary service behind the client. In this case, clients are the point of contact for the message exchange with the server. It holds that every client has a unique IP address such that servers would be required to block all incoming traffic. But, in the notary mode, servers can enforce Cross-Origin Resource Sharing (CORS) policies to prevent communication with external services in the same session.

4.1.2. Insights & Challenges

All three-party TLS oracles commonly rely on secure computation building blocks (PETs in the form of ZKP and MPC systems) [11], [12]. The MPC building blocks are used to set up a secret-shared, client-side TLS session between the oracle verifier and the client (cf. grey and red boxes in Figure 4.3). Subsequently, the client can only proceed according to the TLS specification if the client interacts with the verifier using secure 2PC (cf. blue boxes in Figure 4.3). The 2PC setting gives the verifier the guarantee that the client preserves integrity according to the TLS specification. Further, the client cannot mount MITM attacks as the client only maintains a single secret share of TLS session parameters. MITM attacks are a concern in TLS oracles. With the option to successfully mount MITM attacks, clients know all secret TLS session parameters and remain unnoticed when breaking TLS data integrity. Breaking TLS integrity means that clients can obtain untrue attestations on arbitrary TLS data. Before the external verifier attests TLS data, the verifier imposes a ZKP challenge on the client (cf. yellow boxes in Figure 4.3). In this challenge, clients must convince external verifiers

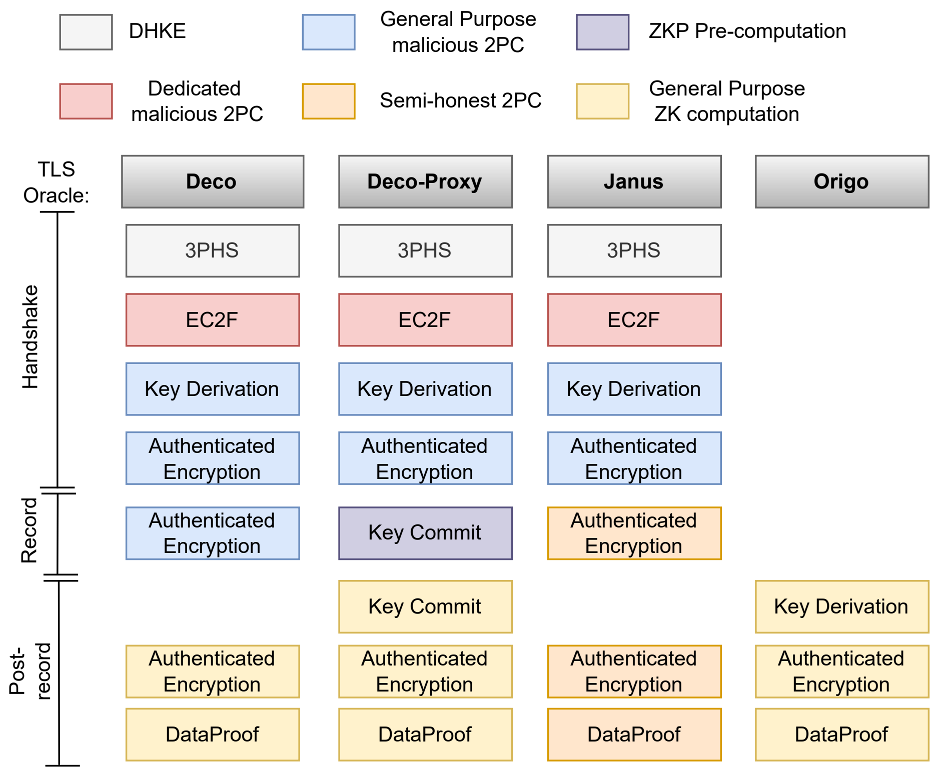


Figure 4.3.: Overview of secure computation building blocks in TLS oracles.

of statements that hold on TLS data. Further, clients are challenged to prove the integrity of TLS data according to a valid TLS session via parameter matches through authenticated encryption algorithms.

In order to improve existing approaches, we pursue two directions which either optimize or remove secure computation building blocks. Leading to our first insight, the example of removing the reliance of PET building blocks exists in the work [12], which proposes an TLS oracle in the proxy mode. One main characteristic of this proxy mode is that it removes the reliance on costly maliciously secure computation techniques in the TLS record phase (cf. Deco-Proxy vs. Deco in Figure 4.3). This optimization comes at the cost of relying on an additional security assumption that prevents malicious clients from mounting MITM attacks. Thus, as our first insight, we focus on the investigation of TLS oracles under different security assumptions in order to remove costly secure computation techniques. Our contribution, called *Origo* (cf. Figure 4.3), leverages an additional security assumption to entirely remove costly 2PC overheads.

As our second insight, we notice that, in the three-party session, the distribution of secret shares changes during the protocol execution. We take this insight as a starting point to investigate the challenge of enhancing the efficiency of PET building blocks. Our contribution, called *Janus* (cf. Figure 4.3), achieves two efficiency gains by tailoring PET building blocks to the oracle-specific parameter distribution.

4.2. *Janus*: Fast Privacy-preserving Data Provenance for TLS

Web users can gather data from secure endpoints and demonstrate the provenance of sensitive data to any third party by using privacy-preserving TLS oracles. In practice, privacy-preserving TLS oracles remain limited and cannot verify larger, sensitive data sets. With this contribution[1], we introduce new optimizations for TLS oracles, which enhance the efficiency of selectively verifying the provenance of confidential web data. The novelty of our work is a construction that secures an honest verifier zero-knowledge proof system in the asymmetric privacy setting while retaining security against malicious adversaries. Concerning TLS 1.3 in the one round-trip time (1-RTT) mode, we propose a new, optimized garble-then-prove paradigm in a security setting with malicious adversaries. Our garble-then-prove paradigm uses semi-honest secure computations for the majority of computations in the garble phase. Our performance improvements reach new efficiency scales in verifying the provenance of private TLS data and facilitate a practical deployment of privacy-preserving TLS oracles in web browsers.

Motivation: In the current age of the Internet where generative Artificial intelligence (AI) boosts the spread of misinformation as never before, industry-leading companies combat misinformation with new data provenance initiatives to maintain a responsible and verifiable data economy [113], [114]. The goal of the initiatives is the establishment and integration of data provenance solutions into today's web, which lacks support for verifiable data provenance. For instance, secure channel protocols such as TLS provide confidential and authenticated communication sessions between two parties: a client and a server. However, if clients present data of a TLS session to any third party (e.g. website), then the third party cannot verify if the presented data originated from an *authentic* and *correct* TLS session (cf. top left part of Figure 4.1). Thus, the third party cannot verify the provenance of the TLS data. In the eyes of the third party, TLS data counts as *authentic* if the origin of the data can be verified. Further, TLS data counts as *correct* if the third party can verify the integrity of presented TLS data against a valid TLS session.

To save a third party from individually verifying data provenance, current approaches either require *servers* to attest to TLS data via digital signatures [50], [110], or employ TLS oracles [11], [12]. Data attestation through *servers* is an efficient data provenance solution but requires server-side software changes and access to a certification infrastructure. By contrast, TLS oracles relieve *servers* from maintaining a data provenance infrastructure by taking over the provisioning and verification of data provenance. Due to the seamless integration into the web, TLS oracles count as *legacy-compatible* as they do not introduce any server-side changes. TLS oracles depend on a *verifier* to examine the provenance of TLS data (cf. oracle verifier in the right part of Figure 4.1). To validate the provenance of TLS data, the *verifier* captures the transcript of a TLS session and challenges the TLS client with a proof computation. If a TLS client can prove *authenticity* and *correctness* of secret TLS session parameters against the

[1]Major parts of this Section 4.2 are subject to copyright protection: Creative Commons Attribution 4.0 International, with permission from [Lauinger, Ernstberger, Finkenzeller, Steinhorst, Janus: Fast privacy-preserving data provenance for TLS, The 25th Privacy Enhancing Technologies Symposium (PETS25), July/2025] (cf. Appendix D).

captured TLS transcript at the *verifier*, then the *verifier* certifies the TLS data of the client. With the certificate, TLS clients are able to convince any third party of data provenance if the third party trusts the *verifier*.

TLS oracles have originated in the context of blockchain ecosystems, where TLS oracles originally solved the "oracle problem" of importing trustworthy data feeds to isolated smart contracts [11], [12], [111]. However, TLS oracles are generally applicable to the Internet, which makes them a crucial technique to build user-centric and data-sovereign systems [84]. For instance, through TLS oracles, users are able to present solvency checks without giving up control and privacy of their data [115]. The accountability and credibility guarantees of data provenance systems are used to combat price discrimination [12], bootstrap legacy credentials [94], or attest if a digital resource originated from a generative AI website [116].

Challenge: Even though different solutions exist, TLS oracles remain constrained in the amount of sensitive data they can validate. This means that for larger sensitive resources such as confidential documents, images, or data sets, data provenance solutions are impractical. For instance, clients are required to prove non-algebraic encryption algorithms (e.g. AES128) in zkSNARK proof systems [12]. However, current zkSNARK proof systems operate efficiently if the computed algorithm relies on algebraic structures (e.g. MiMC [99]). Another approach leverages the structure of TLS 1.3 stream ciphers and separates non-algebraic algorithms from the computations performed by the zkSNARK proof system [11]. In this case, the *client* is required to know the structure of TLS data in advance and cannot selectively verify dedicated parts of TLS records. Even if the computation of non-algebraic algorithms is shifted into a pre-computation phase [57], End-to-end (E2E) efficiency of private provenance solutions remains expensive.

Contribution: Our work addresses the above-mentioned limitations with two new contributions. We leverage the fact that, in the challenge phase (cf. stage 2 in right part of Figure 4.1), TLS oracles introduce an asymmetric privacy setting between collaboratively acting parties; the TLS *client* and the *verifier*. We leverage the asymmetric privacy setting to construct a HVZK proof system with security against malicious adversaries. Our construction relies on a new validation phase which is unilaterally performed by the *client*. The benefit of the HVZK proof system [70] is that it efficiently evaluates non-algebraic algorithms and improves prove computation benchmarks in the challenge phase. Our approach does not require a trusted setup security assumption. With that, our work achieves new E2E benchmarks and solves a main bottleneck of current TLS oracles; the efficient evaluation of legacy algorithms without compromising on security guarantees. Our first contribution applies to TLS oracles running TLS 1.2 or 1.3.

Our second contribution applies to TLS 1.3 in the 1-RTT mode. Here, we require the *client* to select a cipher suite that is supported by the *server*. In a non-optimistic scenario, the *client* is supposed to perform one pre-fetch call. If the *client* sends a compliant CH message during the TLS 1.3 handshake, then the *server* instantly responds with the entire *server*-side handshake transcript. We leverage this effect and show that the *verifier* can securely authenticate the SHTS in a malicious security setting. With access to an authentic SHTS at the *verifier*, we run the garble-then-prove paradigm [117] and rely on a semi-honest 2PC system which does not depend on authenticated garbling. We detect malicious activities of a *client* by matching transcript commitments against authenticity guarantees derived from SHTS. We achieve performance advantages by utilizing more lightweight, semi-honest 2PC for the majority of TLS 1.3 computations.

Result: Our E2E benchmarks for TLS 1.3 verify 8 kB of public TLS data in 0.58 seconds and verify 8 kB of sensitive TLS data 6.7 seconds. Running TLS 1.2, we verify 8 kB of sensitive TLS data 6.2 seconds. Concerning proof computations in the client challenge, our work outperforms related approaches by a factor of $8x$ (cf. Section 4.2.4) and relies on a security setting that does not require a trusted setup assumption. In analogy to Roman mythology, we name these contributions for TLS oracles after the god of transitions, *Janus*. With that, the *Janus* optimizations guard an efficient transition of web resources into a representation where provenance can be verified. In summary,

- We formalize the asymmetric privacy setting of TLS 1.2 and TLS 1.3 oracles. We show that in the asymmetric privacy setting, maliciously secure proof systems can be replaced with a construction that combines a HVZK proof system with a new unilateral validation phase.
- We optimize the efficiency of TLS 1.3 oracles by considering SHTS authenticity guarantees during the garble-then-prove paradigm while retaining security properties equivalent to previous works.
- We analyse the security of our constructions (cf. Appendix A.3), provide performance benchmarks (cf. Section 4.2.4), and open-source[2] the implementation of our secure computation building blocks.

4.2.1. System Model

The system model defines system roles, the threat model, and system goals in form of security properties.

General Notations

The TLS notations of this work are introduced in Section 2.2.3 and closely follow the notations of the work [118]. Further, we denote vectors as bold characters $\mathbf{x} = [x_1, \ldots, x_n]$, where $len(\mathbf{x}) = n$ returns the length of the vector. Base points of elliptic curves are represented by $G \in EC(\mathbb{F}_p)$, where the finite field $\mathbb{F}$ is of a prime size p. For elliptic curve elements, the operators $\cdot, +$ refer to the scalar multiplication and addition of elliptic curve points $P \in EC(\mathbb{F}_p)$. The symbol λ indicates the security parameter. For bits or bit strings, the operators $\cdot$ represent the logical AND, and $\oplus$ represents the logical XOR. Other operators describe a random assignment of a variable with $\xleftarrow{\$}$, the concatenation of strings with $||$, and the comparison of variables with $\stackrel{?}{=}$. Concerning AEAD algorithms in the GCM mode, the

[2]https://github.com/jplaui/janus_artifacts/

symbol $M_{\mathbf{H}}$ is a Galois field (GF) multiplication which translates bit strings into GF(2^{128}) polynomials, multiplies the polynomials modulo the field size, and translates the polynomial back to the bit string representation.

System Roles & Adversarial Behavior

Clients establish a TLS session with *servers*, query data from *servers*, and present TLS data proofs to the *verifiers*. We assume that *clients* behave maliciously and arbitrarily deviate from the protocol specification in order to learn secret shares of TLS session parameters from the *verifier*. Further, malicious *clients* try to learn any information that contributes to convincing the *verifier* of false statements on presented TLS data.

Servers participate in TLS sessions with *clients* and return record data upon the reception of compliant API queries. We assume honest *servers* that follow the protocol specification.

Verifiers act as proxies and take over the role of TLS oracle verifier. *Verifiers* are configured at the client and route TLS traffic between the *client* and the *server*. We assume malicious *verifiers* deviating from the protocol specification with the goal to learn TLS session secret shares or private session data of *clients*.

Threat Model

We rely on a threat model with secure TLS communication channels between *clients* and *servers* (TLS security guarantees hold). Further, we assume that fresh randomness is used per TLS session. Network traffic, even if it is intercepted via a MITM attack by the adversary (e.g. the *client*), cannot be blocked indefinitely. We assume up-to-date DNS records at the *verifier* such that the *verifier* can resolve and connect to correct IP addresses of *servers*. The IP address of a *server* cannot be compromised by the adversary such that adversaries cannot request malicious PKI certificates for a valid DNS mapping between a domain and a *server* IP address. *Servers* share valid PKI certificates for authenticity verification in the TLS handshake phase. Server impersonation attacks are infeasible because secret keys, which correspond to exchanged PKI certificates, are never leaked to adversaries. Our protocol imposes multiple verification checks on the *client* and the *verifier*, where failing verification leads to protocol aborts at the respective parties. All system roles are computationally bounded and learn message sizes of TLS transcript data. For employed ZKP systems, we expect completeness, soundness, and HVZK to hold. We assume that the *client* and *verifier* do not collude.

System Goals

The following security properties concern the *client* and *verifier* as the *server* is assumed to behave honestly.

Session-authenticity guarantees that *verifiers* attests web traffic that originates from an authentic TLS session. Authenticity is guaranteed if the *verifier* successfully verifies the PKI certificate of the server.

Session-integrity guarantees that a malicious *client* and *verifier* cannot deviate from the TLS specification if a TLS session has been authenticated. This means that an adversary cannot modify server-side or client-side TLS traffic in any TLS phase. Notice that for client-side TLS traffic of the record phase, a malicious *client* is able to send arbitrary queries to the *server*, such that *servers* decide if queries conform with API handlers.

Session-confidentiality guarantees that the *verifier* neither learns any TLS session secrets nor any record data that has been exchanged between the *client* and the *server*. Further, the notion guarantees that the *verifier* learns nothing beyond the fact that a statement on TLS record data is true or false.

MITM-resistance guarantees that the properties of *session-integrity*, *session-authenticity*, and *session-confidentiality* hold in a system setting, where adversaries are capable of mounting MITM attacks.

4.2.2. Optimizing Proof Computations In The Asymmetric Privacy Setting

In the following subsections, we analyze TLS oracles with regard to (i) performance tradeoffs and (ii) the asymmetric privacy setting. Further, we secure a HVZK proof system in the asymmetric privacy setting against malicious adversaries. Our construction relies on a new secure validation phase which is unilaterally performed by the *client* (cf. Section A.3.1).

Analyzing Oracles & Asymmetric Privacy

In this section, we analyze the performance bottlenecks of TLS oracles and identify conditions of asymmetric privacy.

Three-party Handshake: TLS oracles turn the two-party protocol of TLS into a three-party protocol by introducing a *verifier* [11]. The *verifier* ensures that the TLS data of the *client* preserves integrity according to an authenticated TLS session via a verifiable computation trace. To audit the integrity of TLS data, the *verifier* and *client* establish a mutually vetting but collaborative TLS client. To construct a collaborative TLS client, TLS oracles replace the TLS handshake with a 3PHS [11], [12]. In the 3PHS, every party injects secret randomness such that the DHE secret on the client side depends on two secrets. As such, the DHE value, which is individually derived at the *server*, can be jointly reconstructed if the *client* and *verifier* add shared secrets together. Appendix 2.2.3 presents the cryptography of the 3PHS.

The consequence of the 3PHS is that the *client* depends on the computational interaction with the *verifier* to proceed in a TLS session with the *server*. The *client* preserves computational integrity according to the TLS specification if the joint TLS computations with the *verifier* progress. Without access to the secret share of the *verifier*, *clients* cannot derive and use full TLS secrets and encryption keys that are required for the secure session with the *server*. Introducing false session parameters on the client side leads to a TLS session abort at the server.

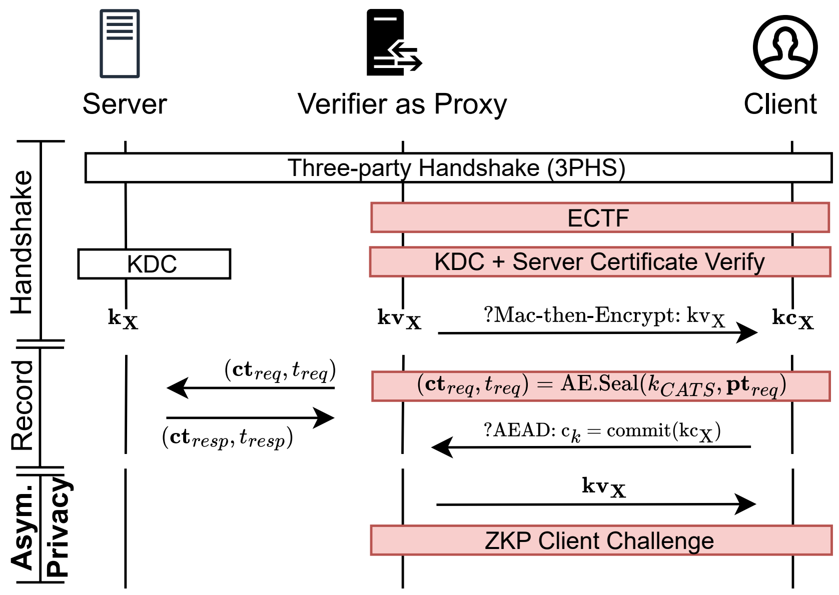

Server
Verifier as Proxy
Client
Handshake
Record
Asym.
Privacy
Three-party Handshake (3PHS)
ECTF
KDC
KDC + Server Certificate Verify
$\mathbf{k}_\mathbf{X}$
$\mathbf{kv}_\mathbf{X}$
?Mac-then-Encrypt: kv_X
$\mathbf{kc}_\mathbf{X}$
$(\mathbf{ct}_{req}, t_{req})$
$(\mathbf{ct}_{req}, t_{req}) = \mathrm{AE.Seal}(k_{CATS}, \mathbf{pt}_{req})$
$(\mathbf{ct}_{resp}, t_{resp})$
?AEAD: $\mathrm{c}_k = \mathrm{commit}(\mathrm{kc}_\mathrm{X})$
$\mathbf{kv}_\mathbf{X}$
ZKP Client Challenge

Figure 4.4.: High-level protocol of TLS oracles. After the key derivation computation (KDC), it holds that $k_X = kv_X + kc_X$, where X indicates server or client side AE keys. Algorithms executed by two parties are surrounded by red boxes and achieve security against malicious adversaries. The syntax '?encryption_mode:' on arrows applies the arrow if the TLS oracle configured the questioned encryption mode.

Client-side Two-party Computation: With secret-shared TLS parameters, the *client* and *verifier* proceed according to the TLS specification by using secure 2PC techniques [12], [119]. To achieve efficient secure 2PC [11], [12], TLS oracles convert secret-shared DHE values in the form of EC coordinates into bit-wise additive secret shares with the ECTF algorithm (cf. Section 2.2.3) [47]. Additive secret shares can be efficiently added together in 2PC circuits that are based on boolean GCs [12], [76], [79], [80], [120]. After the ECTF conversion (cf. Figure 4.4), the *client* and *verifier* perform the TLS key derivation and record phase computations using maliciously secure 2PC based on boolean GCs, which comes with optimized binary circuits for the required computations (e.g. AES) [71], [121]. The general-purpose 2PC computations in TLS are determined by the cipher suite configuration of TLS. Further, existing cipher suite configurations of TLS can be classified according to the encryption mode that is used by the cipher suite algorithms. We enumerate 2PC computations of TLS oracles according to encryption modes as the type of encryption mode impacts the 2PC circuit complexity.

1. **Mac-then-Encrypt (e.g. CBC-HMAC):** The efficiency of TLS oracles in the record phase heavily depends on the cipher suite configuration. If TLS uses Mac-then-Encrypt (MtE) AE (TLS 1.2 with Cipher Block Chaining Hash-based Message Authentication Code (CBC-HMAC)), then the *client* and *verifier* end up deriving four secret-shared keys in the handshake phase:
   - k_{CATS}=kv_{CATS}+kc_{CATS} to encrypt request data $\mathbf{pt}_{req}$.

- k_{SATS}=kv_{SATS}+kc_{SATS} to encrypt response data $\mathbf{pt}_{resp}$.
- k^t_{CATS}=kv^t_{CATS}+kc^t_{CATS} to authenticate requests $\mathbf{t}_{req}$.
- k^t_{SATS}=kv^t_{SATS}+kc^t_{SATS} to authenticate responses $\mathbf{t}_{resp}$.

The *verifier* can disclose the encryption keys kv_{SATS}, kv_{CATS} to the *client*. However, key shares to compute authentication tags must be kept private in order to control the integrity of joint TLS computations [12]. Partly disclosing key shares improves the efficiency of 2PC computations in the record phase. Because *clients* can encrypt or decrypt records locally without costly 2PC calls. Concerning the computation of MtE authentication tags, the 2PC complexity remains independent of record sizes (e.g. using a 2PC-optimized evaluation of HMAC) [12].

Another benefit of CBC-HMAC is that it counts as key committing [12], [56], which guarantees the existence of an unambiguous mapping between a TLS session key and record data. As a consequence, capturing **ct** is the only requirement for the *verifier* before secret shares can be disclosed to the *client*. TLS oracles use the key committing property and simplify the ZKP computation during the client challenge to (i) three invocations of AES and (ii) a selective data opening which leverages the Merkle–Damgård construction [12], [97].

2. **AEAD (e.g GCM / CHACHA20_POLY1305):** If TLS is configured to protect records with AEAD algorithms (TLS 1.3 and optionally TLS 1.2), then the *client* and *verifier* derive two secret-shared keys (k_{CATS}=kv_{CATS}+kc_{CATS}, k_{SATS}=kv_{SATS}+kc_{SATS}). Thus, to maintain *session-integrity*, the *verifier* cannot disclose any secret shared AEAD key in the record phase before receiving a commitment. Since keys are not decoupled as with CBC-HMAC, the efficiency of TLS oracles running AEAD cipher suites deteriorates for larger record sizes. The bottleneck is the 2PC computation of authentication tags, which evaluate algebraic structures (e.g. polynomials over large fields GF(2^{128}) for GCM and GF($2^{130} - 5$) for POLY1305) over all ciphertext chunks.

   Insight 1: The performance of 2PC **AEAD tag computations deteriorates for larger record sizes.**

   In Section 4.2.3, we optimize the 2PC complexity for TLS oracles running AEAD cipher suites in the record phase and our optimization applies to TLS 1.3 in the 1-RTT mode.

   Further, AEAD configurations require special attention as AEAD cipher suites are not key committing [56], [57], [122]–[124]. This means that an adversary can perform commitment attacks [125]. For example, the message franking attack finds two messages $m_1 \neq m_2$ and two keys $k_1 \neq k_2$ such that encrypting m_1 under k_1 and encrypting m_2 under k_2 yield the same ciphertext ct and tag t [126]. This attack is problematic and would break *session-integrity* and, with that, *session-authenticity*. In other words, a successful attack allows the *client* to prove arbitrary TLS data as TLS-authentic in the client challenge. TLS oracles solve this attack by letting the *client* disclose a commitment of the key share to the *verifier* (cf. Figure 4.4) [12]. The extra commitment binds the *client* to a fixed key share, which is verified during the client challenge. Fixing the key

```
$\mathcal{C}_{\text{MtE}}(\mathbf{pt}, kc, kc^t \;;\; kv, kv^t, \mathbf{ct}_\alpha\text{=}\mathbf{ct}\text{[end-3:]}, \phi)$:
1. $t' = \text{HMAC}(kc^t + kv^t, \mathbf{pt})$
2. $\mathbf{ct}_\alpha' = \text{AES}(kc + kv, t')$
3. assert: $\mathbf{ct}_\alpha \overset{?}{=} \mathbf{ct}_\alpha'$, $1 \overset{?}{=} f_\phi(\mathbf{pt})$
$\mathcal{C}_{\text{AEAD}}(\mathbf{pt}, kc \;;\; kv, \mathbf{ct}, \text{iv}, t_\alpha, c_k, \phi)$:
1. $t_\alpha' = [\ \text{AES}(kc + kv, \mathbf{0})\ ,\ \text{AES}(kc + kv, \text{iv} \,||\, \mathbf{1})\ ]$
2. $\mathbf{ct}' = \text{AES}(kc + kv, \mathbf{pt})$; $c_k' = \text{commit}(kc)$
3. assert: $t_\alpha \overset{?}{=} t_\alpha'$, $c_k \overset{?}{=} c_k'$, $\mathbf{ct} \overset{?}{=} \mathbf{ct}'$, $1 \overset{?}{=} f_\phi(\mathbf{pt})$
```

Figure 4.5.: Circuit logic of ZKPs in the client challenge. The semicolon ; separates private inputs (left side) from public inputs (right side). The function f_ϕ evaluates conditions expressed by a public statement ϕ on the plaintext **pt**.

share must happen before the *verifier* discloses remaining session secrets (e.g. $\mathbf{kv_X}$ in Figure 4.4). Otherwise, an attacker can arbitrarily compute valid authentication tags and ciphertext chunks, which is a prerequisite to perform the attack [126].

Client Challenge in Asymmetric Privacy Setting: Once the *client* has gathered enough TLS data, the *verifier* reveals remaining secret shares to the *client* (cf. Figure 4.4). When the *client* obtains full access to session secrets, an asymmetric privacy setting between the *client* and *verifier* is established. Now, the *client* is able to access TLS data by decrypting exchanged records which the *verifier* cannot.

To preserve *session-integrity*, the *verifier* confronts the client with irreversible challenges via ZKP circuits (cf. Figure 4.5). For cipher suites running MtE, the *client* must prove that the plaintext evaluates against the authentication tag which is encrypted under the last three ciphertext chunks. For AEAD cipher suites, the *client* shows that the secret key share *kc* (i) maps to the previously shared key share commitment, (ii) connects plaintext and ciphertext chunks, and (iii) evaluates to intermediate values t_α for the tag computation. Here, the *verifier* validates public ZKP inputs (e.g. t_α) out-of-circuit, which have been shared by the *client*.

Current TLS oracles rely on proof systems (e.g. *Groth16*), which efficiently evaluate algebraic or zkSNARK-friendly arithmetic [12], [57], [97], [117], [119]. However, the ZKP circuits of TLS oracles (cf. Figure 4.5) heavily depend on legacy algorithms (e.g. AES or SHA256) which rely on zkSNARK non-friendly, non-algebraic arithmetic.

Insight 2: Proof systems are not tailored to the arithmetic requirements and the privacy setting found in TLS oracles.

In the following, we secure lightweight proof systems, which efficiently evaluate non-algebraic algorithms, against malicious actors leveraging the conditions found in the asymmetric privacy setting.

HVZK and Asymmetric Privacy

This section picks up on our second insight (cf. Section 4.2.2) and formalizes asymmetric privacy. In the asymmetric privacy setting, we secure a HVZK proof system, which efficiently proves non-algebraic statements, against malicious adversaries. Subsequently, we show how our formal definitions apply to TLS.

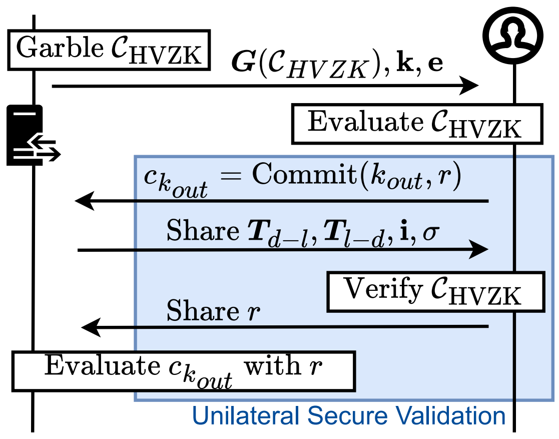
Formalizing Asymmetric Privacy: In the scope of this work, we formalize asymmetric privacy in a setting with three parties; parties p_1 and p_2 and a trusted dealer d. We rely on a maliciously secure 2PC scheme Π_{2PC}, a secure commitment scheme Π_{Com}, and a secret sharing scheme Π_{SS} (cf. Section 2.2.2).

To set up an asymmetric privacy setting between p_1 and p_2, the dealer d calls Π_{SS}.**Share** and individually shares r_1 with p_1 and r_2 with p_2. It holds that the secret shares r_1+r_2 sum to r. We define two cases to commit a message string m into a commitment string c using r. The first case requires p_1 and p_2 to execute a circuit $\mathcal{C}$ in the 2PC scheme Π_{2PC}, where $\mathcal{C}$ calls Π_{SS}.**Reconstruct** and Π_{Com}.**Commit**. In this case, p_1 inputs m and r_1 and p_2 inputs r_2. After the commitment c has been computed and disclosed, p_2 releases the secret share r_2 to p_1 and, with that, initiates the asymmetric privacy setting. Now, p_1 can reconstruct r. With access to m and r, only p_1 is capable of successfully proving Π_{Com}.**Open**.

For the second case, the trusted dealer computes and discloses the commitment string c on a message string m with randomness r. If the trusted dealer performs the commitment, then the dealer additionally shares the message string m with a party (e.g. with p_1). To set up the asymmetric privacy setting, p_2 discloses the secret share r_2 after receiving the commitment string c from the dealer. In the second case, the dealer and p_1 have access to r and can prove a successful commitment opening to p_2.

HVZK and Selective-failure Attacks: To improve the performance of proof computations during the client challenge (cf. Figure 4.4), we deploy a HVZK proof challenge to evaluate the circuits of Figure 4.5. We consider the asymmetric privacy setting between p_1 as the *client* and p_2 as the *verifier*, where p_1 has access to all TLS session secrets. The proof system of the work [70] uses semi-honest 2PC based on boolean garbled circuits to achieve the notion of HVZK and assumes an honest *verifier* (cf. Section 2.2.2). However, in a setting with malicious adversaries, semi-honest 2PC is susceptible to selective failure attacks [79]. Notice that if a malicious p_2 intentionally corrupts one or multiple rows of the garbling tables, p_2 can learn information on which row has been evaluated by p_1. On top and with knowledge of the row permutations, p_2 is capable of deriving secret information of p_1's inputs. In the following subsection, we introduce a secure validation protocol that is unilaterally performed by p_1. The validation detects a maliciously acting p_2 before any secrecy leakage occurs.

Unilateral Secure Validation: The unilateral secure validation is performed once p_1 obtains all public semi-honest 2PC parameters of the HVZK proof system [70], which comprise garbled tables $G(\mathcal{C}_{\text{HVZK}})$ and external labels $\mathbf{e}$ (cf. Section 2.2.3). The parties p_2 and p_1



Garble $\mathcal{C}_{\text{HVZK}}$
$\boldsymbol{G}(\mathcal{C}_{HVZK}), \mathbf{k}, \mathbf{e}$
Evaluate $\mathcal{C}_{\text{HVZK}}$
$c_{k_{out}} = \text{Commit}(k_{out}, r)$
Share $\boldsymbol{T}_{d-l}, \boldsymbol{T}_{l-d}, \mathbf{i}, \sigma$
Verify $\mathcal{C}_{\text{HVZK}}$
Share r
Evaluate $c_{k_{out}}$ with r
Unilateral Secure Validation

Figure 4.6.: Secure unilateral validation protocol in the asymmetric privacy setting to assert correct garbling of $\mathcal{C}_{\text{HVZK}}$.

exchange wire keys **k** corresponding to the private inputs **pt** via the OT_2^1 oblivious transfer protocol [48] and p_2 omits sharing the output label decoding table $\boldsymbol{T}_{l-d}$. The party p_2, acting as the garbler and *verifier*, is convinced of the HVZK proof if p_1, acting as the evaluator and *client*, returns the output wire key that corresponds to the output bit 1. Depending on the cipher suite, the 2PC circuit $\mathcal{C}_{\text{HVZK}}$ implements the logic of the circuits $\mathcal{C}_{\text{MtE}}$ or $\mathcal{C}_{\text{AEAD}}$ (cf. Figure 4.5) and yields a 1 if all assertions are satisfied.

In the default HVZK proof system [70], the *client* p_1 must return the output wire key back to the *verifier* p_2 to complete the HVZK proof protocol. However, to achieve security in a malicious setting, we require p_1 to run a new secure validation phase (cf. Figure 4.6). The unilateral validation enforces p_1 to share a commitment $c_{k_{out}}$ of the output wire key. After sharing the commitment $c_{k_{out}}$, p_2 discloses all garbling parameters of the semi-honest 2PC computation with p_1. Revealing all garbling parameters allows p_1 to verify if $\mathcal{C}_{HVZK}$ has been garbled correctly by recomputing the garbled circuit. And, due to the asymmetric privacy setting, p_1 learns nothing new because all TLS session secrets of p_2 have already been shared with p_1. If p_1 detects a malicious garbling, then p_1 aborts the protocol. Otherwise, p_1 discloses the commitment randomness r such that p_2 can verify the correct output wire key via $c_{k_{out}}$. We show the security of this construction in the Appendix A.3.1.

TLS Compatibility: Our formalization is compatible with the typical TLS oracle setting with a single *verifier*. The *server* takes over the role of the trusted dealer to set up multiplicative secret shares between the client parties via the 3PHS. Subsequently, the ECTF protocol converts client secret shares into an additive representation. The *client* and *verifier* collaboratively commit to TLS session parameters by computing authentication tags and ciphertext chunks (cf. first case commit in Section 4.2.2). Otherwise, the client-side parties receive commitments by capturing server-side traffic (cf. second case commit in Section 4.2.2). Remember that if a cipher suite is not key-committing (e.g. AEAD cipher suites), then the *verifier* needs an additional key share commitment from the *client*. Access to secure commitments is a

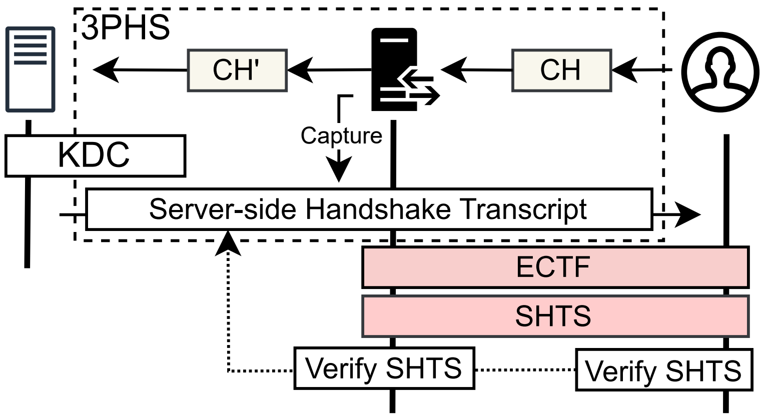


Figure 4.7.: Mutual SHTS verification at client parties. Red boxes indicate values derived in maliciously secure 2PC protocols.

prerequisite for the asymmetric privacy setting. Next, the *verifier* initiates the asymmetric privacy setting by disclosing secret shares of TLS parameters to the *client*. From here on, only the *client* is capable of computing valid commitment openings. Thus, in the client challenge, a HVZK proof system with our unilateral validation protocol can be deployed.

4.2.3. Optimizing End-to-end Performance

Our second contribution applies to TLS oracles running the TLS 1.3 1-RTT mode and targets our first insight (cf. Section 4.2.2) by mainly optimizing 2PC computations in the record phase. In detail, we show how client parties can securely derive and authenticate the SHTS parameter in a malicious security setting. Subsequently, we leverage the SHTS authenticity to deploy an optimized garble-then-prove paradigm, which entirely relies on semi-honest 2PC techniques.

Authenticating SHTS

This section explains why pre-fetching cipher suites are a necessity to reliably validate SHTS authenticity. Further, we show how our SHTS validation sequence counters possible attacks.

Pre-fetch for Immediate Server-side Handshake Transcript: We consider the 1-RTT mode of TLS 1.3, where *servers* immediately derive session secrets and return authenticated handshake messages upon the reception of compliant CH messages (cf. Figure 4.7). Even though TLS 1.3 allows the configuration of three AEAD cipher suites and two possible parameters for the key agreement (ECDHE with X25519 or P-256), *clients* may select an unsupported parameterization. In this case, TLS parameters must be renegotiated. To prevent any renegotiation, we expect *clients* to perform a single pre-fetch call to detect possible configurations. This way, *clients* and *verifiers* can reliably expect and capture the server-side handshake transcript if a compliant CH message is sent to the *server*.

Compute and Disclose of SHTS: Once *clients* receive the server-side handshake traffic, both the *client* and *verifier* continue to derive secret-shared session parameters via the 3PHS (cf. Section 2.2.3) and the ECTF protocol (cf. Section 2.2.3) [11], [12]. In the end, the *verifier* locally maintains s_1 and the *client* locally keeps s_2 and it holds that $s_1 + s_2 = \text{DHE}$. To derive SHTS, the *verifier* and *client* evaluate the circuit $\mathcal{C}_{\text{SHTS}}$ (cf. Table 4.1) in a maliciously secure 2PC system. Similar to the works [12], [56], [97], we leverage the fact that, during the handshake phase, the *client* can securely disclose the SHTS parameter to the *verifier*. Even though the *verifier* knows SHTS, the key independence property of TLS 1.3 prevents the *verifier* from learning the HS secret [73], as HS is protected by **hkdf.exp** (cf. line 5 and 17 of Figure 2.5). Without access to HS, the adversary cannot derive application traffic keys from HS.

Attacking SHTS Authenticity: The server-side handshake transcript contains the SF message, which can be seen as a commitment to established TLS session parameters [56], [97]. We require both client parties to capture the server-side handshake transcript before the *client* and *verifier* compute SHTS via the 2PC circuit $\mathcal{C}_{\text{SHTS}}$ (cf. Figure 4.7). This condition prevents adversaries from forging the authenticity of SHTS as client parties can validate handshake session secrets against the commitment (cf. Appendix A.3.2).

To provide more context, the following aspects must be considered. Our system model prevents the adversary (i) from compromising the *server*'s private key and (ii) from accessing full handshake secrets through the 3PHS and the ECTF protocol. Thus, to obtain a valid signature from the *server*, an adversary must replay a previous and individually established handshake transcript. The adversary can take on the following two roles:

1. **Malicious Client:** If the adversary takes the role of the *client*, then the *verifier* injects fresh randomness by determining the CH transcript. As a consequence, a replayed handshake signature does not match the new transcript, and the adversary cannot sign a new transcript without the *server*'s private key. Thus, in this scenario, the signature verification detects malicious behavior.

2. **Malicious Verifier:** If the adversary acts as a *verifier*, then the adversary determines the CH message transcript which the *client* cannot. This gives the adversary the opportunity to replay a previously established handshake session, where the adversary knows the session secret DHE. If we allow the computation of SHTS before capturing the server-side handshake transcript, the following attack is possible[3]: The adversary picks a random input to compute SHTS′ and recomputes the last part of a previously established handshake transcript using SHTS′ (cf. lines 8,9,11 of Figure 2.5). Once the adversary shares the forged server-side handshake transcript, the *client* accepts because the SF validation succeeds. Afterward, the adversary accepts any incoming requests from the *client* which could contain confidential data (e.g. credentials). This attack compromises *session-authenticity* and *session-integrity*. However, if the *client* honestly

[3]This attack applies to the work [12] in the TLS 1.3 mode. The work [119] indicates this attack. Our work provides the first full attack description and proposes a countermeasure.

Table 4.1.: Mapping of TLS Computation traces to 2PC circuits. We use XHTS for SHTS/CHTS and XATS for SATS/CATS.

Circuit	Computation Trace
$\mathcal{C}_{\text{XHTS}}$	DHE=s_1+s_2; DHE to XHTS
$\mathcal{C}_{(\text{k,iv})}$	DHE=s_1+s_2; DHE to ($kc_{\text{XATS}}, kv_{\text{XATS}}, \text{iv}_{\text{XATS}}$)
$\mathcal{C}_{\text{CB}_{2+}}$	($kc_{\text{XATS}}, kv_{\text{XATS}}, \text{iv}_{\text{XATS}}$) to CB$_{2+}$
$\mathcal{C}_t$	($kc_{\text{XATS}}, kv_{\text{XATS}}, \text{iv}_{\text{XATS}}$, ct) to t
$\mathcal{C}_{\text{open}}$	DHE=s_1+s_2; DHE to SHTS; DHE to CB

proceeds the 2PC computations according to TLS 1.3, then *session-confidentiality* on record phase data holds.

We close this attack with the previously defined condition, where client parties capture the server-side handshake transcript before the joint evaluation of SHTS. Here, the adversary is faced with the following challenge. The adversary must replay a handshake transcript that complies with the prospective SHTS value. Since the adversary cannot predict the outcome of $\mathcal{C}_{\text{SHTS}}$ without access to the secret s_2 of the *client*, the adversary has negligible chances of guessing a compliant SF message beforehand (cf. Appendix A.3.2). This way, *session-authenticity* and *session-integrity* hold.

SHTS Validation: The validate SHTS, the *client* and *verifier* match the locally computed SF′ values against the SF value from the *server*. To access SF, both client parties derive the handshake traffic secrets $k_{\text{SHTS}}, \text{iv}_{\text{SHTS}}$ and decrypt server-side handshake messages. Additionally, the client parties assert the validity of the *server*'s certificate. Afterwards, the client-side jointly computes $\mathcal{C}_{\text{CHTS}}$ using maliciously secure 2PC, which outputs the Client Handshake Traffic Secret (CHTS) to the *client*. With CHTS, the *client* completes the handshake by computing and sharing the CF message.

Garble-then-prove with Semi-honest 2PC

We apply a modified garble-then-prove paradigm [117]. In the garble phase, we replace 2PC computations based on authenticated garbling with lightweight semi-honest 2PC computations that do not require authenticated garbling. We show the security of our construction in the Appendix A.3.3.

Intuition of Garble-then-Prove: The idea behind the garble-then-prove paradigm is as follows. If a malicious *client* acts as the garbler of the semi-honest 2PC system, then the *client* can mount selective failure attacks throughout the record phase. However, as TLS oracles eventually disclose session secrets of the *verifier*, the malicious *client* learns nothing beyond what the honest *client* would have learned. The prove phase is supposed to (i) detect any cheating activities of the *client* and (ii) provide the *verifier* with a conditional abort option

```
$\mathcal{C}_{\mathbf{zkOpen}}(s_2, \mathbf{pt}\ ;\ s_1, \mathbf{I}^{\text{open}}, \mathbf{ct}, t, \text{SHTS}, \phi)$:
1. $\text{SHTS}', \mathbf{CB}' = \mathcal{C}_{\text{open}}(s_1, s_2, \mathbf{I}^{\text{open}})$
2. $\mathbf{ct}' = \mathbf{CB}' \oplus \mathbf{pt}$, $t' = [\ \text{CB}_0', \text{CB}_1'\ ]$
3. assert: $\text{SHTS}' \overset{?}{=} \text{SHTS}$, $\mathbf{ct}' \overset{?}{=} \mathbf{ct}$, $t' \overset{?}{=} t$, $1 \overset{?}{=} f_\phi(\mathbf{pt})$
$\mathcal{C}_{\mathbf{tpOpen}}(s_2\ ;\ s_1, \mathbf{I}^{\text{open}}, \mathbf{CB}, t, \text{SHTS})$:
1. $\text{SHTS}', \mathbf{CB}' = \mathcal{C}_{\text{open}}(s_1, s_2, \mathbf{I}^{\text{open}})$, $t' = [\text{CB}_0', \text{CB}_1']$
2. assert: $\text{SHTS}' \overset{?}{=} \text{SHTS}$, $t' \overset{?}{=} t$, $\mathbf{CB} \overset{?}{=} \mathbf{CB}'$
```

Figure 4.8.: Extending the ZKP circuit $\mathcal{C}_{\text{AEAD}}$ (cf. Figure 4.5) to the HVZK circuit used by our E2E-optimized TLS 1.3 oracle for the privacy-preserving and transparent client challenges.

before any data attestations occur. To do so, the proving phase recomputes and compares all semi-honest 2PC computations of the *client* against securely authenticated session parameters at the *verifier* (cf. Section 4.2.3).

Garble Phase: In the garble phase, the *client* and *verifier* collaboratively evaluate multiple 2PC circuits (cf. Table 4.1), where the *client* acts as the garbler and the *verifier* acts as the evaluator. In the handshake phase, the circuit $\mathcal{C}_{(\text{k,iv})}$ yields secret-shared application traffic secrets to the *client* and *verifier*. The additive relation of secrets (e.g. k_{XATS}=kc_{XATS}+kv_{XATS}) continues to hold.

In the record phase, the 2PC circuit $\mathcal{C}_{\text{CB}_{2+}}$ outputs counter blocks CB_i (cf. Figure 2.3) to the *client*, with i>1. To prevent commitment attacks on records, no block CB_i ever includes any CB_0,CB_1 blocks. To encrypt a request, the *client* computes $\mathbf{ct}_{req}$ = $\mathbf{CB}_{2+} \oplus \mathbf{pt}_{req}$ and shares the ciphertext $\mathbf{ct}_{req}$ with the *verifier*. Next, the client parties jointly compute t_{req} using $\mathcal{C}_t$. The *verifier* sends the request to the *server*. After client parties receive a response ($\mathbf{ct}_{resp}$, t_{resp}), the *verifier* discloses all session secrets and initiates the asymmetric privacy setting. Notice that computing the AEAD key commitment c_k is redundant because, in the client challenge, we can consider SHTS as an authenticated commitment on session secrets.

Prove Phase: The proving phase starts with the asymmetric privacy setting, where the *verifier* has captured the TLS 1.3 transcript and disclosed session secrets to the *client*. The prove phase of this work considers the authenticated SHTS parameter as a commitment string. Further, the 2PC circuit $\mathcal{C}_{\text{HVZK}}$ of the client challenge is set to the $\mathcal{C}_{\mathbf{zkOpen}}$ algorithm (cf. Figure 4.8). The circuit $\mathcal{C}_{\text{open}}$ derives SHTS and counter blocks $\mathbf{CB}_i$, where the list of indices $\mathbf{I}^{\text{open}}$ indicates the plaintext/ciphertext chunks of interest. The *client* determines $\mathbf{I}^{\text{open}}$ according to plaintext chunks that are passed to f_ϕ. Notice that the assertion of the authentication tag is reduced to comparing intermediate values CB_0, CB_1. The in-circuit derivation of CB_0, CB_1 hides application traffic keys from the *verifier* and frees $\mathcal{C}_{\mathbf{zkOpen}}$ from expensive algebraic operations (e.g. multiplication of GF polynomials). Remember that the HVZK proof system which evaluates $\mathcal{C}_{\mathbf{zkOpen}}$ runs the unilateral secure validation to detect a malicious *verifier*. After the client challenge, the *verifier* derives t_{resp}' from t' out-of-circuit and asserts if $t_{resp}' \overset{?}{=} t_{resp}$. If all assertions succeed, the *verifier* attests the TLS 1.3 data of the *client*.

Additional Considerations: The following aspects complete the context of TLS oracles beyond our contributions.

1. **Operation Modes:** TLS oracles can be operated in two different modes, which introduce distinct arrangements in the proving phase. Both operation modes depend on a list of indices $\mathbf{I}^{\text{open}}$, which the *client* shares with the *verifier*. The *client* selectively determines $\mathbf{I}^{\text{open}}$ once all session secrets are obtained. Otherwise, the *client* could not access data of server-side records. The *verifier* uses $\mathbf{I}^{\text{open}}$ to identify public inputs in the form of ciphertext chunks for the verification of $\mathcal{C}_{\text{HVZK}}$ (cf. Figure 4.8).

   - **Transparent Mode:** In the *transparent* mode, the *verifier* checks (i) if the AEAD encryption of presented plaintext chunks matches the captured ciphertext transcript and (ii) if a computation trace from the encryption key to SHTS exists. To prevent the *verifier* from learning TLS encryption keys, the *transparent* mode requires an adapted circuit $\mathcal{C}_{\textbf{tpOpen}}$ (cf. Figure 4.8). $\mathcal{C}_{\textbf{tpOpen}}$ takes as public input counter blocks **CB**, which have been computed and shared by the *client*. Further, $\mathcal{C}_{\textbf{tpOpen}}$ asserts SHTS, counter blocks **CB**, and authentication tags via intermediate values. The assertions $1 \overset{?}{=} f_{\phi}(\mathbf{pt})$ and $\mathbf{ct} \overset{?}{=} \mathbf{ct}' = \mathbf{CB} \oplus \mathbf{pt}$ are computed out-of-circuit because plaintext chunks are publicly disclosed.
   - **Privacy-preserving Mode** In the *privacy-preserving* mode, the *client* does not share **pt**. Instead, the *client* shares $\mathbf{I}^{\text{open}}$ and proves knowledge of authentic plaintext data via the HVZK proof system. To do so, the *client* evaluates the 2PC circuit $\mathcal{C}_{\textbf{zkOpen}}$ and applies the unilateral secure validation.

   Example: We assume that TLS is configured to use AES in the GCM mode. Further, we assume that the TLS data of interest for the validation according to f_{ϕ} is contained in ct_3 of the response (**ct**, t). In this case, the index i=3 is included in the list $\mathbf{I}^{\text{open}}$. With $\mathbf{I}^{\text{open}}$, the **zkOpen** circuit is able to compute the right $\text{CB}_{2+\text{index}}$, and consider $\text{CB}_5 = \text{AES}(k, \text{iv}||\ldots 5)$ for the computation of $ct_5' = \text{CB}_5 \oplus pt_5$. If the assertions against public inputs succeed (e.g. $ct_5' \overset{?}{=} ct_5$), and CB_5 has been derived with secrets that match a verified SHTS, then the data pt_5 preserves *session-integrity* and *session-authenticity*.

2. **Processing Multiple Records:** Concerning the collaborative processing of multiple records in the *record phase*, we differentiate computations with respect to the following dependencies:

   - **Requests are independent of responses.** If no request depends on the contents of a response, then the circuit $\mathcal{C}_{\text{CB}_{2+}}$ is only called for the compilation of requests. Response CBs can be locally computed by the *client* once the asymmetric privacy setting enforces the disclosure of full session secrets to the *client*.

- **Requests depend on responses.** If a request of number $n > 1$ depends on the contents of responses $\mathbf{ct}$=$[ct_1, \ldots, ct_l]$, where each response ct_m has an index $m < n$, then the *client* and *verifier* perform l executions of the circuit $\mathcal{C}_{\text{CB}_{2+}}$. The evaluation of l circuits $\mathcal{C}_{\text{CB}_{2+}}$ yields l vectors of encrypted counter blocks $\mathbf{CB}_{2+}$ to the *client*. With l vectors of $\mathbf{CB}_{2+}$, the *client* is capable of accessing the contents of the responses $\mathbf{ct}$=$[ct_1, \ldots, ct_l]$ to construct the n-th request. To preserve *MITM-resistance* and prevent commitment attacks, it must hold that the *verifier* intercepts the pair $(\mathbf{ct}, t)$ before the circuit $\mathcal{C}_{\text{CB}_{2+}}$ outputs the corresponding $\mathbf{CB}_{2+}$ of response $\mathbf{ct}$.

3. **Data Attestation:**

   If the client challenge succeeds successfully, the *verifier* attests to the verified TLS data. The attestation structure depends on the operation modes. In the transparent mode, the *verifier* hashes verified TLS data and signs the hash. Thus, the certification parameter p_{cert}=(t, ϕ, pk, σ) of the transparent attestation includes a signature σ=**ds.Sign**(sk,[ϕ, t]) computed at time t. The *verifier* overwrites the statement $\phi = H(\mathbf{pt})$ to the hash of verified data such that every third party can evaluate presented TLS data against ϕ and against arbitrary statements. In the privacy-preserving mode, the structure of p_{cert} remains the same except that the statement ϕ expresses asserted constraints. The privacy-preserving attestation convinces any third party of the fact that the *verifier* successfully validated TLS data provenance against the statement ϕ at time t. The certificate p_{cert} enables verifiable TLS data provenance as p_{cert} can be verified by any third party who trusts the *verifier* and the *server*.

4.2.4. Evaluation

The evaluation describes the software stack and measures the impacts of our two optimizations. The first optimization improves proof computation times for TLS 1.2/1.3 oracles. The second optimization improves the E2E performance of TLS 1.3 oracles. We provide micro benchmarks on a circuit level in Appendix B.1.1. Initially, we provide a cipher suite analysis to justify our choice of cipher suite implementation.

Cipher Suite Analysis

TLS allows parties to select multiple cipher suites. In order to implement secure computation circuits for the most used cipher suite, we perform a cipher suite analysis.

To evaluate cipher suite support among today's APIs, we scanned the first 15k entries of the `top-1m.csv.zip` list[4]. To perform the scan, we rely on a publicly available TLS cipher suite scanner[5]. We remove scans that encounter network errors (e.g. no such host) or TLS errors (e.g. EOF, handshake failures). The cipher suite support distribution is depicted in Figure 4.9. TLS 1.2 configured with GCM reaches a support of 73.5% while TLS 1.2 CBC-HMAC reaches

[4] `https://github.com/PeterDaveHello/top-1m-domains`

[5] https://github.com/TeoLj/TLSscanner

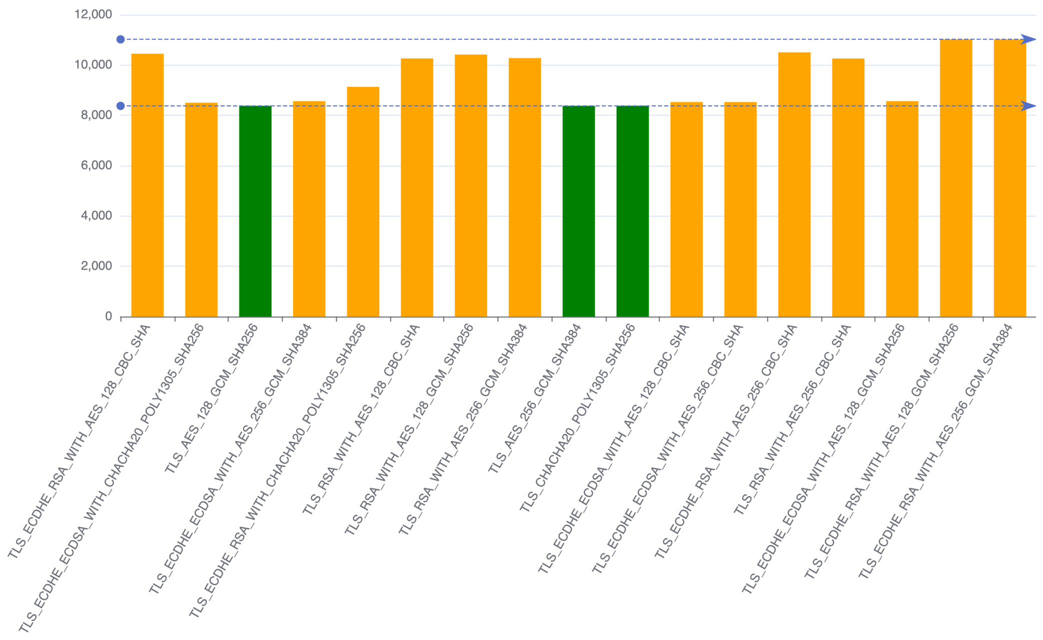


Figure 4.9.: TLS cipher suite scan performed at the 11th of June 2024. Green bars refer to TLS 1.3 cipher suites and yellow bars indicate TLS 1.2 cipher suites.

70.05%. TLS 1.3 support is lower at 55.8%. Even though TLS oracles relying on CBC-HMAC have efficient record phase computations, multiple attacks on the CBC vMAC-then-encrypt pattern have been introduced [127]–[129]. Even though countermeasures exist, protecting records with the CBC vMAC-then-encrypt pattern is not recommended anymore [130]. Our distribution of scans aligns with this recommendation. Further, most endpoints support AEAD cipher suites, where the TLS 1.2 support is 17.7% ahead of TLS 1.3.

Implementation

Tooling: We implement the 3PHS by modifying the Golang *crypto/tls* standard library[6] and configure the NIST P-256 elliptic curve for the ECDHE. Our ECTF conversion algorithm uses the Golang Paillier cryptosystem[7]. We use the *mpc* library[8] to access semi-honest 2PC based on garbled circuits, which supports the optimizations free-XOR [131], fixed-key AES garbling (AES-NI instruction set) [132], and half-gates [133]. We adjust *mpc* to output single wire labels if we execute 2PC circuits in the context of the HVZK proof systems. We rely on the *ag2pc* framework[9] to implement maliciously secure 2PC circuits in TLS 1.2. To compute ZKPs, we rely on the *gnark* framework [134]. We open-source our secure computation circuits[10].

[6] `https://pkg.go.dev/crypto/tls`
[7] `https://github.com/didiercrunch/paillier`
[8] `https://github.com/markkurossi/mpc`
[9] `https://github.com/emp-toolkit/emp-ag2pc`
[10] `https://github.com/jplaui/circuits_janus`

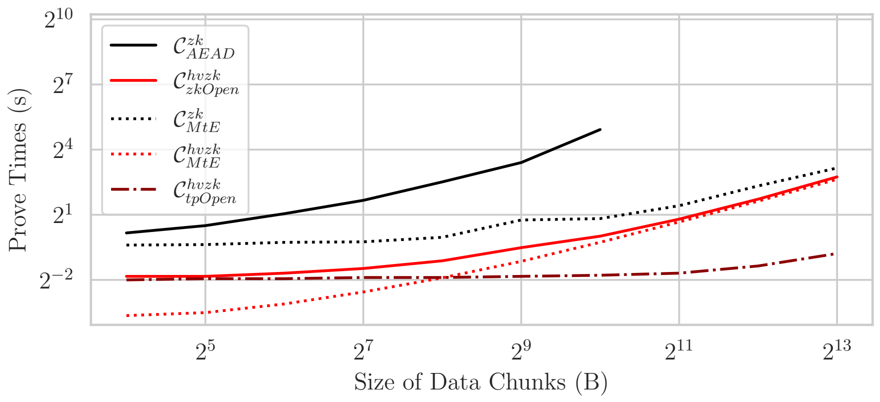


Figure 4.10.: Scalability analysis of ZKP circuits, where circuits $\mathcal{C}_{MtE}$ (dotted) are compatible with TLS 1.2 only. Circuits $\mathcal{C}^{hvzk}$ leverage *Janus* optimizations. Lines closer to the bottom right corner are "better" and prove more data in less time.

Based on our cipher suite analysis, we implement the secure computation circuits AES128 and SHA256 to achieve compatibility with the most popular TLS 1.2/1.3 cipher suites.

E2E Benchmarks: The numbers of *TLSn*[11] and *DiStefano*[12] are reproduced by running publicly available experiments (cf. Table 4.3). Due to the fact that Deco [12] is closed source, we open source our $Deco_{zk}$ re-implementation[13], which executes TLS 1.2 configured with CBC-HMAC. The implementation of *Janus12* is equal to $Deco_{zk}$ except for the post-record phase. Here, *Janus12* employs the HVZK proof system implemented with the *mpc* framework. The TLS 1.3 oracles *DecoProxy* and *Janus13* rely on AEAD cipher suites which we implement with the *mpc* library. Malicious 2PC circuits computed with the *mpc* framework use the dual-execution mode [80]. We rely on *gnark* to implement ZKP circuits for *Deco*, *DecoProxy*, and *Origo*.

Performance

All performance benchmarks have been averaged over ten executions and have been collected on a MacBook Pro configured with the Apple M1 Pro chip and 32 GB of RAM.

Client Challenge Benchmarks: Concerning our first optimization, we evaluate ZKP circuits that are used during the client challenge. We execute the traditional circuits $\mathcal{C}^{zk}_{AEAD}$, $\mathcal{C}^{zk}_{MtE}$ (cf. Figure 4.5) as a baseline using the fastest *gnark* proof system *Groth16*. We execute the circuits $\mathcal{C}^{hvzk}_{zkOpen}$ as an AEAD variant with the SHTS assertion (cf. Figure 4.8) and $\mathcal{C}^{hvzk}_{MtE}$ using the HVZK proof system. We depict the protocol support of the circuit variants in Table 4.2. Generally, the HVZK circuits (cf. red in Figure 4.10) achieve the best performance, where MtE-based circuits are ahead of AEAD circuits. This makes sense as the circuit $\mathcal{C}^{hvzk}_{zkOpen}$ requires additional logic of constant size to derive and validate keys against SHTS. For larger

[11] https://github.com/tlsnotary/tlsn/tree/main
[12] https://github.com/brave-experiments/DiStefano/tree/main
[13] https://github.com/jplaui/decoTls12MtE

Table 4.2.: Mapping protocols to cipher suites which support the configuration of the suite of algorithms ECDHE_ECDSA_AES128_SHA256.

Mode	Variant	Protocols
`12_GCM`	AEAD	*TLSn*
`12_CBC`	MtE	*Deco,Janus12*
`13_GCM`	AEAD	*DiStefano*, *DecoProxy*, *Origo*, *Janus13*

data sizes, this overhead diminishes as AES (e.g. AEAD circuits) or SHA256 (e.g. MtE circuits) dominate the circuit complexity. Further, we benchmark the transparent validation of TLS data via $\mathcal{C}_{tpOpen}$ (cf. Figure 4.5), which outperforms the privacy-preserving validation for data sizes beyond 500 bytes.

Optimized End-to-end Performance: We present E2E benchmarks of open-source TLS oracles in Table 4.3. Concerning TLS 1.2 (cf. top part of Table 4.3), we run *Janus12* using CBC-HMAC to benefit from constant size circuits in the record phase. For *TLSn*, which runs TLS 1.2 using AEAD, record phase 2PC complexity is determined by the size of the record (cf. row 1 vs row 2/3 in Table 4.3). Even though the post-record communication increases, *Janus12* achieves the fastest post-record execution benchmarks. For instance, the HVZK proof computations of *Janus12* outperform related works by a factor of 9 in the Local Area Network (LAN) setting.

Concerning TLS 1.3, *Origo* and *DecoProxy* circumvent 2PC computation of the record phase by introducing an additional trust assumption (*clients* cannot mount MITM attacks). As a result, these works behave equal to the TLS 1.3 baseline in the record phase. *DiStefano* sets the fastest handshake execution times, which we link to the enhanced MtA algorithm in the ECTF protocol [119], [135]. Our implementation does not incorporate handshake optimizations proposed by *DiStefano* yet, because we achieve practical handshake benchmarks with our Paillier-based MtA conversion of session secrets. *Janus13* runs an AEAD cipher suite and evaluates 8 times more data (256b request, 2kb response) while remaining practical in all protocol phases.

4.2.5. Discussion

The discussion presents related works and summarizes remaining limitations and future work directions. We summarize our feature analysis of related works in Table 4.4.

Related Works

The work *XYWY23* [117] introduces the garble-then-prove paradigm based on semi-honest 2PC with authenticated garbling. After the garble phase, authenticated garbling bits are transformed into a Pedersen commitment which can be opened in a zkSNARK proof system. By contrast, our E2E optimization for TLS 1.3 derives SHTS authenticity in a malicious setting. In the garble phase, we deploy semi-honest 2PC without authenticated garbling. To compute

Table 4.3.: End-to-end (E2E) benchmarks of open-sourced TLS oracles. For the LAN setting, we assume a round-trip-time RTT=0 ms and a transmission rate r$_t$=1 Gbps. The Wide Area Network (WAN) setting assumes a Round-trip Time (RTT) =50 ms and r$_t$=100 Mbps. Protocols starting with an * require an additional security assumption. Works marked with a ′ use a transparent setup assumption to compute ZKPs. Entries marked with " take over the value of the entry above.

			Communication (kb)				**Execution LAN (s) / WAN (s)**			
Protocol	**Data**	**vTLS**	Offline	Handshake	Record	Post-record	Offline	Handshake	Record	Post-record
TLS	-	1.2	-	1.6	0.67	-	- / -	0.32 / 0.72	0.3 (ms) / 0.2	- / -
$\mathit{TLSn}_{\text{tp}}$	579b	1.2	178 (MB)	36 (MB)	40 (MB)	0.57	3.3 / 17.54	1.18 / 4.46	1.12 / 4.52	0.76 / 0.9
$\mathit{Deco}_{\text{zk}}$	32b	1.2	523.51 (MB)	294.52	150.55	0.23	14.38 / 56.26	0.94 / 2.36	0.68 / 1.59	0.76 / 0.86
$'\mathit{Janus12}_{\text{zk}}$	32b	1.2	415.22 (MB)	"	"	28.13	2.9 / 36.1	" / "	" / "	0.08 / 0.23
TLS	-	1.3	-	1.42	0.71	-	- / -	0.36 / 0.76	0.49 (ms) / 0.2	- / -
$^*\mathit{Origo}_{\text{zk}}$	32b	1.3	367.61 (MB)	"	"	0.24	29.16 / 58.56	" / "	" / "	1.26 / 1.36
$^*\mathit{DecoProxy}_{\text{zk}}$	32b	1.3	578.47 (MB)	307.7	"	0.27	24.34 / 70.61	0.95 / 2.37	" / "	1.35 / 1.45
DiStefano	256b	1.3	220.484 (MB)	343.42	48.82	-	5.85 / 23.48	0.43 / 0.85	0.12 / 0.32	- / -
$'\mathit{Janus13}_{\text{tp}}$	2.2kb	1.3	305.17 (MB)	113.8	984	583	1.99 / 26.4	0.51 / 0.91	1.04 / 1.51	0.46 / 0.6
$'\mathit{Janus13}_{\text{zk}}$	2.2kb	1.3	406.29 (MB)	"	"	2 (MB)	2.63 / 35.13	" / "	" / "	2.08 / 2.34

Table 4.4.: Related works feature comparison.

Paper	**Type**	**Key Feature**
TLSn/Deco	Notary	3PHS & 2PC-based TLS Client
Zombie	Proxy	Pad Commit
Origo	Proxy	2PC-free
DiStefano	Notary	Secure MtA & Browsing Privacy
XYWY23	Notary	Garble-then-prove & Pedersen Commit
Janus	Proxy	Secure 2PC-based hvZKP

proofs efficiently, our work relies on a 2PC-based HVZK proof system with a unilateral validation phase.

The work *Zombie* [57] notices that legacy algorithms constitute over 40% of TLS computations and decouples stream cipher computations with a *pad commitment*. The *pad commitment* is used to partly outsource legacy algorithms from the ZKP circuit to a pre-processing phase. We tackle the arithmetic requirements of legacy algorithms with a well-suited HVZK proof system.

The work *DiStefano* [119] secures a leakage of the Paillier-based MtA conversion protocol [136] and achieves browsing privacy, where the *verifier* does not lean which *servers* the *client* queries.

Our contribution *Origo* [97] proposes a 2PC-free oracle solution and reduces 2PC bandwidth overheads at the cost of requiring an additional security assumption (*clients* cannot mount MITM attacks).

Another way to improve the efficiency of the *client* challenge is to decouple the maliciously secure 2PC evaluation of CBs, which is done in the works *DiStefano*, *TLSn*, *Deco* [11], [12], [119]. Notice that this optimization applies to TLS oracles which run AEAD cipher suites. Here, the *client* obtains output wire keys and shares a commitment of CB wire keys with the *verifier*. With the commitment, the *verifier* discloses the wire key decoding table as well as secret shares to the *client*. The *client* is now able to verify the correctness of $\mathcal{C}_{\text{CB}_{2+}}$, access response data, and select a transparent data opening. Optionally, *clients* can prove TLS data in a ZKP circuit which (i) takes in private output wire keys, (ii) computes CBs with the decoding table as public input, and (iii) authenticates TLS data by XORing a plaintext with CBs to the intercepted ciphertext. This approach has the following limitations. The wire key possession before obtaining a decoding table prevents the *client* from accessing response data such that the *client* remains with two options. With knowledge of the plaintext structure, the client commits to a selection of output encodings, which correspond to the CBs of interest for the privacy-preserving data opening. Without knowledge of the plaintext structure, the client uses a Merkle tree commitment structure to commit to all output encodings and selectively opens CBs in the ZKP circuit via Merkle tree inclusion proofs [11]. Due to frequent updates, API data is unlikely to remain static over a longer period of time such that the scenario of not knowing plaintext structures prevails. Our work, in contrast, allows clients to selectively prove plaintext data during the client challenge.

Limitations

Reliable executions of *Janus* optimizations for TLS 1.3 oracles requires an additional pre-fetch call. We recommend that operators of the *verifier* pre-fetch *servers* before launching Oracle sessions. This way, *verifiers* can provide seamless attestation services for *clients*. We like to highlight that the *verifier* as a proxy is susceptible to blocking if server-side endpoints detect reoccurring Transmission Control Protocol (TCP) sessions from the same proxy IP [57], [137]. In this case, we recommend running the *verifier* as a notary service behind the *client* (as described by *TLSn*, *Deco* [11], [12]). Our work does not investigate compression techniques to lower 2PC bandwidth overheads. Neither does our work currently support the MtA optimizations pointed out by *DiStefano* [119] (1. ring-based MtA instead of Paillier-based MtA, 2. parallelized MtA for TLS 1.3 authentication tags). We consider these topics for future improvements.

Disclaimer: Legal and Compliance Issues

This section informs users and companies running the *Janus* TLS oracle about subsidiary conditions and agreements. As TLS oracles are *legacy-compatible*, companies running the *verifier* connect seamlessly to web endpoints which are queried by users. Web endpoints do not necessarily notice the *verifier*. Legal issues (e.g. copyright infringements) arise if users export proprietary content or declare false data ownership. In this case, companies must deny content. If companies operate oracles in the privacy-preserving mode, then companies learn nothing from transport data beyond the statement validity. In the transparent mode, companies can surveil opened data in plain. Users must be aware that companies learn network layer data (e.g. IP addresses, domains), which is required to operate the *proxy* service. Tracking or profiling oracle data may cause regulatory compliance violations.

Asymmetric Privacy & Related Concepts

Our definition of asymmetric privacy relates to the concept of a Trapdoor hash function (THF) between two parties [138]. THF guarantees function privacy for the sender and input privacy for the receiver. The private function evaluates receiver data at a private index. In contrast, our asymmetric privacy setting ensures input privacy for a sender and convinces the receiver of a public function that has been evaluated on the entire sender input.

Further, to differentiate against other notions such as asymmetric differential privacy [139], our notion of asymmetric privacy targets a threshold number of parties with access to commitment secrets.

Applications

Generally, the *Janus* optimizations make ZKP-computing *clients* practical in constrained environments (e.g. browsers, mobile). And, with that, serve existing Oracle applications such as confidential financial instruments, legacy credentials, or the combating of price discrimination [12]. On top, our scalability benefits open new application fields where larger data sets or documents require proofs of provenance. In this context, our contributions help in fighting the dissemination of disinformation by attesting generative AI content, which is among the goals of the Coalition for Content Provenance and Authenticity (C2PA) [113], [140].

4.3. *Origo*: Bandwidth-optimized Data Provenance

TLS oracles enable content authenticity and integrity beyond the two-party TLS session. As such, *clients* can present a verifiable notion of TLS data to external parties which are not part of the TLS session. With our contribution *Origo* [97], we remove any 2PC computation dependence of TLS oracles and, with that, drastically improve bandwidth requirements. Our optimization relies on a weaker network adversary that is assumed to be incapable of mounting MITM attacks between the server and the Oracle verifier.

Motivation: TLS oracles operate in a three-party setting, where an additional *verifier* attests to the integrity and authenticity of TLS data. Before any attestation occurs, the *verifier* takes part as an additional client in the TLS session. The collaborative TLS client side is constructed using MPC techniques [11], [12]. MPC techniques ensure that, instead of having full access to TLS session secrets, the *verifier* and client only keep a secret shared part of session secrets. This way, the *verifier* is able to witness the integrity of TLS conform computation by the client without ever getting access to full session secrets. Without full access to session secrets, the *verifier* cannot learn which data is exchanged between the client and the server. As a consequence, in order to attest valid data, the *verifier* challenges the client via a ZKP computation. If the assertions made by the ZKP convince the *verifier*, then the *verifier* attests TLS data from the client.

Challenge: Even though the efficiency of PET techniques in TLS oracles has been steadily improved, PET computations continue to introduce costly overheads, especially concerning the network bandwidth requirements (e.g. caused by maliciously secure 2PC). As a result, no scalable deployment of TLS oracles exists in today's web. Our work[14] takes on the challenge to investigate if the network bandwidth overheads can be lowered or removed entirely. To do so, we consider TLS 1.3 oracle in the proxy mode and rely on an additional security assumption. Similar to the work [12] in the proxy mode, we assume a weaker network adversary who is incapable of performing MITM attacks between the server and the *verifier*. However, instead of applying this network adversary in the record phase only, our work investigates if an earlier deployment is possible. Additionally, we analyze the commitment properties of TLS 1.3. Here, the goal is to identify how many security guarantees can be established by extending the ZKP circuit during the client challenge.

Contribution: We notice that in the above-described setting, a malicious *verifier* cannot replay a previously established TLS session in order to trick the client into accepting a new TLS session. The reason is that every new session at the client introduces fresh randomness. This leads to a different set of encryption keys and the *verifier* cannot forge a signature that matches the new session transcripts. This way, the client can detect a malicious verifier during the verification of the server certificate. On the other hand, a malicious verifier does not succeed in convincing the *verifier* of false TLS data because we propose a new ZKP circuit that remediates commitment attacks on TLS cipher suites (e.g. the message franking attack on the

[14]Major parts of this Section 4.3 are subject to copyright protection: Creative Commons Attribution 4.0 International, with permission from [Ernstberger*, Lauinger*, Wu, Gervais, Steinhorst, ORIGO: Proving Provenance of Sensitive Data with Constant Communication, The 25th Privacy Enhancing Technologies Symposium (PETS25), July/2025] (cf. Appendix D).

GCM encryption mode [126]). As such, *Origo* tailors a ZKP circuit to the conditions found in TLS 1.3 in the 1-RTT mode and provides verifiable provenance of TLS data with constant communication complexity. *Origo* establishes itself as an interesting candidate for TLS oracles operating in wide area networks (WANs) or for TLS oracles with bandwidth-constrained, client-side environments. In summary, our contributions concerning the work *Origo* in this thesis are as follows:

- We describe the *Origo* protocol by extending the system model of the work *Janus*. We put special emphasis on the description of the *Origo* ZKP circuit which includes optimizations at the SHACAL-2 level.
- We disclose an entire end-to-end implementation of the *Origo* protocol, including ZKP circuits[15].
- We benchmark the end-to-end protocol execution of *Origo*

4.3.1. Extended System Model

The work *Origo* uses the same system roles which we define in the *Janus* system model (cf. Section 4.2.1). This means that the *Origo* oracle *verifier* operates in the proxy mode as well. Equally, all *Janus* system goals hold for the work *Origo*. In the *Origo* protocol, the system goal of *MITM-resistance* can only be achieved with an additional security assumption which we define in a different threat model.

Threat Model

Again, we assume secure TLS channels between servers and clients and rely on fresh CH key share randomness injected by clients. Network traffic cannot be intercepted by clients. Even though this is a strong security assumption, we highlight that the detection of Border Gateway Protocol (BGP) hijacking attacks, which lead to MITM attacks [141], can be prevented by active monitoring techniques [142]. In *Origo*, we expect completeness, soundness, and zero-knowledge to hold for the employed ZKP system. Equally as in the threat model of our contribution *Janus*, we assume up-to-date DNS records and honest servers that never share private keys that are associated with the server-side PKI certificates. System roles are computationally bounded and learn the sizes of exchanged messages. Last, the client and Oracle verifier cannot collude.

4.3.2. *Origo* Protocol

This section introduces the *Origo* protocol according to the illustration of Figure 4.11 with respect to different TLS phases. *Origo* works in the three-party setting of TLS oracles, where the *verifier* eventually attests to TLS data.

[15]`https://github.com/jplaui/origo`

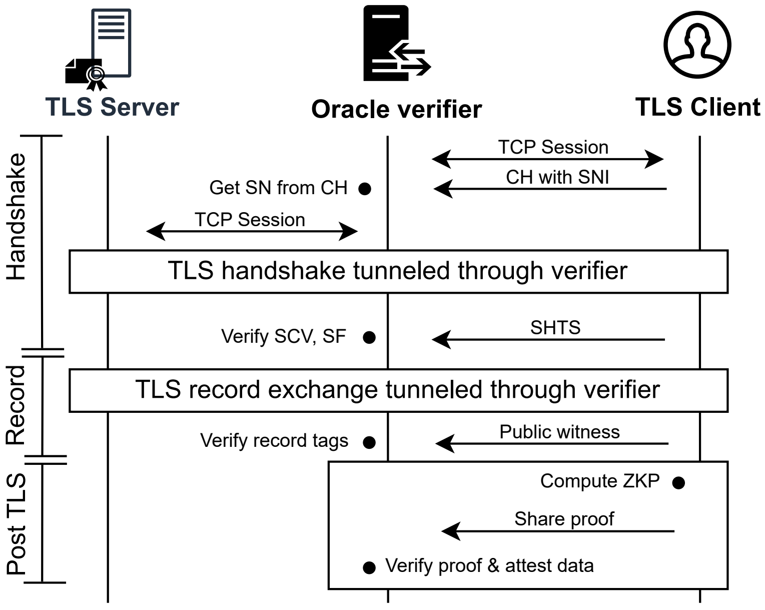


Figure 4.11.: Illustration of the *Origo* protocol with respect to the TLS 1.3 protocol phases. Besides message exchanges, the figure highlights additional processing requirements at the *verifier* and the *client*.

Handshake Phase

Initially, the client sets up a TCP connection with the verifier who operates as a proxy. Next, in contrast to related works [11], [12], [119], no 3PHS is performed. Instead, the *client* sets up a TCP connection to the *verifier* and uses the Server Name Identifier (SNI) extension of TLS to communicate the domain of the server (destination address) via the CH message. The *verifier* parses the CH message, extracts the server name, and resolves the server's IP address using DNS. Afterward, the *verifier* can successfully set up a TCP session with the server in order to tunnel a dual message exchange using both TCP sessions (client session and server session). Right after both TCP connections exist, the *verifier* forwards the CH message to instantiate the TLS handshake between the client and the server (cf. handshake phase of Figure 4.11).

After the TLS handshake, the *verifier* has access to the handshake message transcript. To complete the TLS session agreement, the *client*, among other values, derives the SHTS secret, which, due to the TLS 1.3 key independence property, can be securely disclosed to the *verifier* [73]. With SHTS, the *verifier* is able to decrypt the server-side handshake traffic. This gives the verifier access to the server's PKI certificate and the server-finished (SF) digest. The next task at the *verifier* is the verification of the SF digest and the PKI certificate. If the validation fails, the *verifier* aborts the protocol. Otherwise, the *client* proceeds with the TLS record phase.

```
Origo ZKP Circuit ($\text{dHS}^{\text{in}}$, $\text{pt}^i$ ; $\mathbf{H}_0$, $\mathbf{H}_2$, $\mathbf{H}_3$, $\mathbf{SHTS}^{\text{in}}$, $\mathbf{h}^{\text{HS,opad}}$, $\mathbf{MS}^{in}$, $\mathbf{SATS}^{\text{in}}$, $\mathbf{SATK}^{\text{in}}$, $\mathbf{CB}^i_0$, $\mathbf{CB}^i_1$, $\mathbf{sIV}^i$, $\mathbf{ct}^i$, $\phi$) :
1. $tk_{\text{sapp}}$, $\text{SHTS}^{\text{in}} \leftarrow \text{KDC}(\text{dHS}^{\text{in}}, \mathbf{H}_0, \mathbf{H}_2, \mathbf{H}_3, \mathbf{h}^{\text{HS,opad}}, \mathbf{MS}^{in}, \mathbf{SATS}^{\text{in}}, \mathbf{SATK}^{\text{in}})$
2. $\text{ct}^i \leftarrow \text{AES128\_GCM}(\mathbf{sIV}^i, \text{pt}^i, tk_{\text{sapp}})$
3. $\mathbf{CB}^i_0, \mathbf{CB}^i_1 \leftarrow \text{AES128}(\mathbf{sIV}^i, tk_{\text{sapp}})$
4. Assert if $\text{ct}^i \overset{?}{=} \mathbf{ct}^i$ and if $\text{CB}^i_0 \overset{?}{=} \mathbf{CB}^i_0$ and if $\text{CB}^i_1 \overset{?}{=} \mathbf{CB}^i_1$ and if $\text{SHTS}^{\text{in}} \overset{?}{=} \mathbf{SHTS}^{\text{in}}$
5. Assert if $1 \overset{?}{=} f_\phi(\text{pt}^i)$
```

Figure 4.12.: Circuit logic of the *Origo* ZKP proof. Function arguments behind the semicolon ; indicate the public witness.

Record Phase

In the record phase, the *client* and server start to exchange application-specific data. Again, the task of the *verifier* is to capture the record phase TLS transcript and forward exchanged messages of the TLS parties. Once the *client* has collected the data to be attested by the *verifier*, then the *client* discloses a pair of intermediate AEAD parameters ($\text{CB}^i_0, \text{CB}^i_1$) per record r_i. Notice that the index i selects record data contributing to the conviction of the *verifier* in the ZKP challenge. If record data at indices k does not contribute to proving a statement in the ZKP challenge, then the records r_k require no further processing. The *verifier* uses the all pairs ($\text{CB}^i_0, \text{CB}^i_1$) and the captured ciphertext chunks ($\mathbf{ct}^i_j$) to compute TLS 1.3 message authentication tags t^i. Subsequently, the *verifier* matches recomputed tags t^i with captured message authentication tags. If the tag verification fails, then the *verifier* aborts the protocol. Otherwise, the parameters $\text{CB}^i_0, \text{CB}^i_1$ can be considered authentic and used as public inputs to the ZKP computation in the post TLS phase.

In addition to the counter blocks $\text{CB}^i_0, \text{CB}^i_1$, the *client* discloses additional parameters to the *verifier*. All values together constitute the public witness which is used for the evaluation of the ZKP circuit (cf. Figure 4.12). Figure 4.15 indicates the remaining out-of-circuit computations that concern the public witness. Here, the lines denoted by the initial character v are executed by the *verifier* and validate the witness parameters against intercepted transcript values. The lines denoted by the initial character p generate public witness parameters and are computed by the *client* who, in the ZKP context, acts as the prover.

Post TLS Computations (Challenge Phase)

The post-record phase executes the ZKP challenge between the *verifier* and the *client*. Here, the ZKP circuit evaluates the logic which is depicted in Figure 4.12. The key derivation computation (KDC) of the ZKP circuit evaluates the lines indicated by zk in the Figure 4.15. The circuit ensures that malicious clients cannot mount commitment attacks on the TLS 1.3 AEAD cipher suites [126]. For example, the commitment attack of message franking finds two keys that encrypt two plaintexts to the same ciphertext and authentication tag (cf. Section 4.2.3). We leverage the observation presented in the work [56], which considers the

SHTS parameter as a server-authenticated commitment. If the key to compute ciphertext chunks evaluates successfully against the intercepted SHTS parameter, then the message franking commitment attack becomes infeasible. Because the ZKP computation fixes the key by evaluating the key against the SHTS commitment. As such, no second key can be used to encrypt a second plaintext against the ciphertext. The *client* can neither present a forged SHTS′ value because the SHTS computation depends on server-side randomness which the *verifier* obtains before the *client*.

Once the ZKP protocol completes successfully, the *verifier* attests the statement ϕ and the evaluated TLS data which has been proven by the *client*. The attestation structure introduced by our contribution *Janus* equally applies to *Origo* (cf. Section 4.2.3). Since *Origo* does not require any two-party MPC computations and the protocol overhead mainly depends on the ZKP computation, we set out to optimize the ZKP circuit further. In the following, we explain multiple optimizations to reduce the complexity of the *Origo* ZKP circuit.

4.3.3. Optimized TLS 1.3 Key Derivation

The following subsections describe parts of the TLS key derivation that must be recomputed in the *Origo* ZKP circuit. Before we outline the ZKP circuit at the level of the SHACAL-2 compression function, we show how and where the ZKP computation of HMAC can be optimized using SHACAL-2.

Key Derivation ZKP Circuit

The key derivation ZKP circuit is a building block used by the proxy to force the client into computing correct and non-ambiguous TLS data provenance proofs. To recap the context of data provenance proofs, two challenges must be checked by the proxy. First, the proxy must verify a valid authentication of the intercepted TLS 1.3 transcript data. In order to authenticate the transcript data, the proxy uses the server's PKI certificate to validate the server's transcript signature. Second, the proxy requires the client to show a non-ambiguous mapping of private TLS 1.3 session keys against intercepted and public TLS 1.3 transcript data. Correctness of data provenance holds because the challenges ensure that the client cannot forge the signature of transcript data. Thus, under the assumption of an honest server, the transcript data is correct. Further, the non-ambiguity of the session keys lets the proxy verify that record layer data proofs comply with a correct TLS 1.3 session.

To verify non-ambiguity and, with that, the correctness of session keys, the proxy demands the client to compute a ZKP circuit. The ZKP circuit ensures the integrity of a cryptographically binding mapping between private session keys and public TLS 1.3 transcript data. A cryptographically binding computation is a collision-resistant function evaluation that guarantees an explicit and unequivocal mapping of input data to a specific output. Additionally, the ZKP circuits maintains the privacy of private session keys. Thus, if the client is able to compute a valid proof of the ZKP circuit, the proxy is convinced that the non-ambiguity of session keys holds.

```
ZKP Circuit: Key Derivation(HS; $\mathbf{H}_2$, $\mathbf{H}_3$, SHTS)
1. SHTS$\leftarrow$ hkdf.exp(HS,"s hs traffic" || $\mathbf{H}_2$)
2. dHS $\leftarrow$ hkdf.exp(HS,"derived", H(""))
3. MS $\leftarrow$ hkdf.ext(dHS, 0)
4. Client Application Traffic Secret (CATS) $\leftarrow$ hkdf.exp(MS, "c ap traffic" || $\mathbf{H}_3$)
5. SATS $\leftarrow$ hkdf.exp(MS, "s ap traffic" || $\mathbf{H}_3$)
6. $tk_{capp}$ $\leftarrow$ DeriveTK(CATS)
7. $tk_{sapp}$ $\leftarrow$ DeriveTK(SATS)

SHTS Verification(SHTS, $\mathbf{H}_7$, SF)
1. $fk_S$ $\leftarrow$ hkdf.exp(SHTS, "finished" || "")
2. SF' $\leftarrow$ HMAC($fk_S$, $\mathbf{H}_7$)
3. SF' $\overset{?}{=}$ SF
4. ok $\overset{?}{=}$ verifyCertificate()
```

Figure 4.13.: The first function in the figure presents the high-level ZKP circuit to prove the TLS 1.3 key derivation. The second function is computed without a ZKP circuit at the proxy to verify the finished authentication tag. We highlight variables which count as public input with **bold** text.

The naive approach to compute a cryptographically binding mapping between private session keys and their public transcript is to follow TLS 1.3 key exchange and key derivation specification. In the TLS 1.3 handshake, the server and client randomly choose secrets $x \overset{\$}{\leftarrow} \{0,1\}^{256}$ and $y \overset{\$}{\leftarrow} \{0,1\}^{256}$, which they exchange based on public diffie-hellman key exchange parameters $\mathbf{X}$=g^x and $\mathbf{Y}$=g^y. The parties involved in the secret exchange obtain a shared secret by computing DHE $\leftarrow \mathbf{Y}^x = \mathbf{X}^y$ with their respective secret randomness. With access to the shared secret DHE, the server and client are eligible to compute the handshake secret HS $\leftarrow$ hkdf.extr(**dES**, DHE), which only they can compute by knowing DHE. The parameter **dES** is publicly known and computes as **dES** $\leftarrow$ hkdf.exp(ES,"derived", H("")) with ES $\leftarrow$ hkdf.extr(0,0). With access to HS, the client and server derive handshake and record layer application traffic secrets. The work [56] shows that deriving HS based on the private input y and public input $\mathbf{Y}$ leads to a non-ambiguous sample of HS. Further, derivation and verification of the server certificate signature must be computed in the ZKP circuit to verify a correct authentication of $\mathbf{Y}$ and, thus, HS. Doing all these computations in the ZKP circuit is costly and luckily, a shortcut exists.

Due to the key independence property of TLS 1.3 [73], the client can disclose the SHTS to the proxy without compromising the security of HS and record layer application traffic secrets. Leaking SHTS is possible because `hkdf.exp` protects HS through a pre-image resistant hash function. TLS 1.3 achieves pre-image resistance since the input to `hkdf.exp` contains sufficient secret randomness. The proxy uses SHTS to decrypt handshake traffic and to verify the server-finished message (cf. SHTS verification in Figure 4.13). Further, the proxy accesses the server's certificate and can efficiently verify transcript authenticity with out-

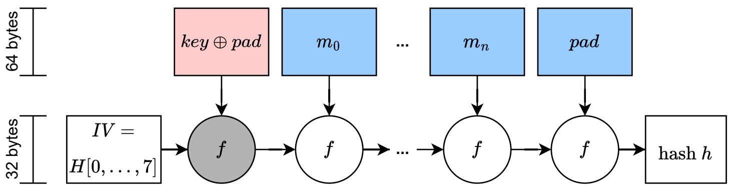


Figure 4.14.: Merkle-Damgård structure of the SHA256 hash function. Values marked in red indicate private input whereas blue background indicates public input. To protect a secret *key*, the prover must compute the first f (grey background) in circuit. All remaining intermediate hash values f (white background) can be computed out of circuit by the verifier and checked against the public input hash value h.

of-circuit computations. We depict the efficient key derivation circuit in Figure 4.13, where we follow the convention of notations introduced in the work [118]. Notice that transcripts hashes $\mathbf{H}_7$ = H(ClientHello||... ||ServerCertVfy), $\mathbf{H}_2$ = H(ClientHello||ServerHello), $\mathbf{H}_3$ = H(ClientHello||... ||ServerFinished), and $\mathbf{H}_0$ = H(“ ”) are computed at the proxy. Further, application traffic secrets are computed according to the formula $tk_{sapp} \mid tk_{capp}$ = (key,iv) = DeriveTK(secret) = (hkdf.exp(secret,“key”,H(“ ”), L_k), hkdf.exp(secret,“iv”,H(“ ”), L_{iv}))), where L_k/L_{iv} indicate the key or iv length of the selected TLS 1.3 cipher suite. In the next section, we show how the ZKP circuit can be optimized further.

HMAC Optimization

The TLS 1.3 key derivation can be further optimized when looking at the structure of HMAC and its underlying hash function SHA256. The optimization is relevant because `hkdf.extr` and `hkdf.exp` both make use of HMAC. Further depending on the required key size of the TLS 1.3 cipher suite, `hkdf.exp` calls HMAC multiple times. By taking the TLS 1.3 cipher suite `TLS_AES_128_GCM_SHA256` as an example, 128-bit encryption keys are used such that both functions `hkdf.extr` and `hkdf.exp` call HMAC once internally.

HMAC is computed according to the Formula 4.1, where $\oplus$ denotes bitwise XOR, || denotes concatenation, and b=64 bytes when using SHA256.

$$\begin{aligned} HMAC(k,m) = &H((K' \oplus opad)||H((K' \oplus ipad)||m)) \\ &\text{with } K' = H(K), \text{ if } len(K) > b \\ &\text{and } K' = K, \text{ else} \end{aligned} \tag{4.1}$$

With Equation 4.1 and the TLS 1.3 cipher suite `TLS_AES_128_GCM_SHA256`, the concatenation of the inner hash $H((K' \oplus ipad)||m)$ (32 bytes) and $K' \oplus opad$ (64 bytes) yields a 96-byte output, which in turn, is the input to the outer hash function. The input to the inner hash function is of size $64 + \text{len}(m)$ bytes. Thus, both hash input sizes in HMAC are above 64 bytes. If the hash input of SHA256 is above 64 bytes, SHA256 applies the Merkle-Damgård structure which repeats calls to an internal compression block cipher f to reduce the input to a

Initialize:

$l_0 = $ “tls13 derived”; $l_2 = $ “tls13 s hs traffic”; $l_3 = $ “tls13 s ap traffic”

$l_f = $ “tls13 finished”; $l_k = $ “tls13 key”; $l_{iv} = $ “tls13 iv”;

$m^{\mathbf{H}_2} = 32 \,||\, len(l_2) \,||\, l_2 \,||\, len(\mathbf{H}_2) \,||\, \mathbf{H}_2 \,||\, 1$; $m^{\mathbf{H}_3} = 32 \,||\, len(l_3) \,||\, l_3 \,||\, len(\mathbf{H}_3) \,||\, \mathbf{H}_3 \,||\, 1$;

$m^{\mathbf{H}_0} = 32 \,||\, len(l_0) \,||\, l_0 \,||\, len(\mathbf{H}_0) \,||\, \mathbf{H}_0 \,||\, 1$; $m_{iv} = 12 \,||\, len(l_{iv}) \,||\, l_{iv} \,||\, 0 \,||\, 1$;

$m_f = 32 \,||\, len(l_f) \,||\, l_f \,||\, 0 \,||\, 1$; $m_k = 16 \,||\, len(l_k) \,||\, l_k \,||\, 0 \,||\, 1$;

Key Derivation Trace:

1. p: $h^{\text{HS},ipad}, l_x = f\,(\text{IV}, 0, \text{HS} \oplus ipad)$; $\mathbf{SHTS}^{in}, \_ = f\,(h^{\text{HS},ipad}, l_x, m^{\mathbf{H}_2})$
2. p: $\mathbf{h}^{\text{HS},opad}, l = f\,(\text{IV}, 0, \text{HS} \oplus opad)$
3. v: $\text{SHTS}\,, \_ = f\,(\mathbf{h}^{\text{HS},opad}, l, \mathbf{SHTS}^{in})$; $fk_S{}^{in}, \_ = f\,(f\,(\text{IV}, 0, \text{SHTS} \oplus ipad), m_f)$
4. v: $fk_S\,, \_ = f\,(f\,(\text{IV}, 0, \text{SHTS} \oplus opad), fk_S{}^{in})$; $\text{SF}'^{in}, \_ = f\,(f\,(\text{IV}, 0, fk_S \oplus ipad), \mathbf{H}_7 \,||\, 1)$
5. v: $\text{SF}'\,, \_ = f\,(f\,(\text{IV}, 0, fk_S \oplus opad), \text{SF}'^{in})$; $\text{SF}' \overset{?}{=} \mathbf{SF}$

10. p: $\text{dHS}^{in}, l = f\,(h^{\text{HS},ipad}, l_x, m^{\mathbf{H}_0})$
11. zk: $\text{dHS}\,, \_ = f\,(\mathbf{h}^{\text{HS},opad}, l, \text{dHS}^{in})$
12. p: $\mathbf{h}^{\text{dHS},ipad}, l = f\,(\text{IV}, 0, \text{dHS} \oplus ipad)$
13. v: $\mathbf{MS}^{in}, \_ = f\,(\mathbf{h}^{\text{dHS},ipad}, l, \text{0bytes})$
14. zk: $\text{MS}\,, \_ = f\,(f\,(\text{IV}, 0, \text{dHS} \oplus opad), \mathbf{MS}^{in})$
15. p: $\mathbf{h}^{\text{MS},ipad}, l = f\,(\text{IV}, 0, \text{MS} \oplus ipad)$
16. v: $\mathbf{SATS}^{in}, \_ = f\,(\mathbf{h}^{\text{MS},ipad}, l, m^{\mathbf{H}_3})$
17. zk: $\text{SATS}\,, \_ = f\,(f\,(\text{IV}, 0, \text{MS} \oplus opad), \mathbf{SATS}^{in})$

18. p: $\mathbf{h}^{\text{SATS},ipad}, l = f\,(\text{IV}, 0, \text{SATS} \oplus ipad)$
19. v: $\mathbf{SATK}^{in}, \_ = f\,(\mathbf{h}^{\text{SATS},ipad}, l, m_k)$
20. zk: $h^{\text{SATS},opad}, l = f\,(\text{IV}, 0, \text{SATS} \oplus opad)$
21. zk: $tk_{sapp}\,, \_ = f\,(h^{\text{SATS},opad}, l, \mathbf{SATK}^{in})[:16]$
22. v: $\mathbf{sIV}^{in}, \_ = f\,(\mathbf{h}^{\text{SATS},ipad}, l, m_{iv})$
23. p: $\mathbf{sIV}, \_ = f\,(h^{\text{SATS},opad}, l, \mathbf{sIV}^{in})[:12]$

Figure 4.15.: Computation trace of the key derivation of a single TLS 1.3 application traffic key. The figure differentiates between in-circuit and out of circuit computations which are separately computed by the prover and verifier respectively. The function f is the one-way compression blockcipher SHACAL-2 of SHA256 and takes a 32 byte input as the first argument, a padding length helper as the second argument, and a 64 byte input as the third argument. The function returns the length l (required for padding) of the hashed input and the output of the SHACAL-2 blockcipher. The initialization vector of f computes as $\text{IV} = H[0, \ldots, 7]$.

fixed-sized output. The compressing block cipher SHACAL-2 of SHA256 uses 64 computation rounds to hide its input and has not been broken [143]. Thus depending on whether the inner or outer hash is computed, the first call of the one-way compression block cipher inside SHA256 already hides inputs $(K \oplus ipad)$ or $(K \oplus opad)$ of size 64 bytes and with that, hides the secret K of the prover [12]. As a result, the output of the compressing block cipher in

SHA256 can be used as public input to reduce ZKP circuit complexity.

Figure 4.14 shows the case which applies in a ZKP circuit to compute the HMAC inner hash $H_{inner} = H((K' \oplus ipad)||m)$, where e.g. $m = \mathbf{H}_2$ is publicly known input. If m is publicly known by the verifier, the prover can compute the grey f and disclose it to the verifier, which computes the remaining part of the hash out of the circuit. The same optimization of SHA256 is feasible when computing the outer hash $H_{\text{outer}} = H((K' \oplus opad)||\mathbf{H}_{\text{inner}})$. Thus, proving HMAC in a ZKP takes two evaluations of the SHACAL-2 compression function f if the message input m is publicly known.

Key Derivation Broken Down

At the level of compression function f calls, Figure 4.15 shows the computation trace of the TLS 1.3 key derivation. The figure summarizes computations executed at the proxy as the verifier (v) and the client as the prover (p, zk), where zk indicates computations that are part of the optimized ZKP circuit. Public parameters are highlighted in bold, which leaves the value dHS^{in} as the only private input of the client. The ZKP circuit shows that if the client discloses $\mathbf{h}^{\text{HS},opad}$ and $\textbf{SHTS}^{in}$ as public input (line 2 and 3), SHTS (line 4) and dHS (line 11) can be computed while non-ambiguously mapping to the only private input dHS^{in} of the optimized ZKP circuit. Thus, in total, deriving either one of the application traffic keys tk_{capp} or tk_{sapp} takes 8 ZKP circuit computations of the SHA256 one-way compression function f (lines 11, 14, 17, 20, 21).

4.3.4. Evaluation

For a distinctive analysis of the security and performance of TLS oracles, we present the *Origo* performance benchmarks first.

Experimental Setup

All experiments are conducted on a MacBook Pro configured with 32 GB of RAM and the Apple M1 Pro chip. All results are averaged over 10 executions. We configured the TLS 1.3 cipher suite `TLS_AES_128_GCM_SHA256` using the Golang TLS standard library[16]. We modified the TLS standard library to parse intermediate transcript values that can be utilized as input parameters to the ZKP circuits of different TLS oracles. We relied on *gnark* [134] for general-purpose ZKPs. We implemented SHA256 and AES128 in gnark to realize *Origo*. We open-source our implementation performance evaluation in a public repository[17].

Evaluation Scenarios

Due to the complexity of TLS oracles, we separate experiments into different scenarios to evaluate the tradeoffs of different TLS oracle types.

[16] `https://pkg.go.dev/crypto/tls`
[17] `https://github.com/jplaui/origo`

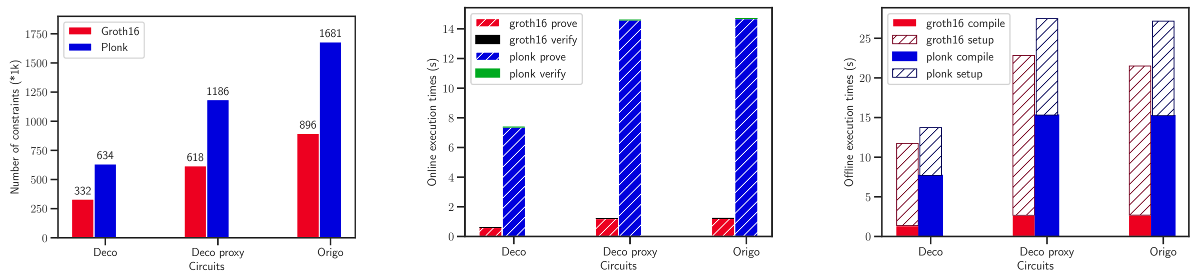


Figure 4.16.: Benchmarks of oracle ZKP circuits executed using the *gnark* framework.

Baseline: The experimental setup considers the execution of the TLS 1.3 protocol as the baseline performance reference. Existing TLS oracles follow the TLS 1.3 baseline configuration but depend on additional 2PC or ZKP communication and computation overheads.

End-to-end Benchmarks: In the first place, we are interested in evaluating system end-to-end times of TLS oracles to answer the question if, from a performance perspective, considering a stronger security assumption is beneficial. For the composition of end-to-end evaluation times, we consider data sizes communicated between the client and the verifier, and execution times in the LAN and WAN setting (cf. Table 4.3). In order to have a fair comparison of end-to-end benchmarks, we evaluate a specific data provenance proof of solvency among all oracle types. Details of ZKP gadgets can be found in the Appendix B.2.1.

Values in the execution columns of Table 4.3 sum execution times conducted by the client and verifier. We consider 2PC online computations as 2PC overhead and ZKP Prove and Verify calls as ZKP overhead. Offline values aggregate all computations that can be pre-computed. For 2PC, offline computation encompasses function-dependent and function-independent pre-computations. Circuit to constraint system compilation times and the ZKP setup function count as ZKP offline execution times. Concerning the LAN and WAN settings, an additional overhead must be considered. This overhead affects two metrics, where the first metric can be represented by the multiplication of the number of client-side messages times the RTT divided by two. The second metric is the transmission time which divides the communicated bytes by the supported bit rate of the network.

ZKP Circuit Benchmarks: We evaluate *Origo*, *Deco*, and *DecoProxy* based on their ZKP circuits. Further, we measure how different proof systems behave with respect to the circuit logic (cf. Figure 4.16). This evaluation answers how much ZKP overhead *Origo* produces.

Discussion

We discuss our evaluation results with regard to the following three questions:
Q1: Does the *Origo* proxy setting, which assumes MITM attack infeasibility for malicious clients, yield any drastic performance benefits, and where?

Answer: The MITM resistance of *Origo* removes reliance on secure 2PC computations. As a result (cf. Table 4.3), *Origo* behaves equally to the TLS baseline in the handshake and record phases. Concerning execution times in the post-record phase, and compared to 2PC oracles, *Origo* tends to have higher ZKP execution times in both the LAN, and the WAN setting. The observations can be explained with the ZKP circuit provided in Figure 4.12, which is more complex due to the additional key derivation circuit. Since 2PC oracles have less complex ZKP circuits, prove times of *Origo* are almost twice as long as prove times of 2PC oracles. However, most of the ZKP computation overhead can be moved to the offline pre-computation phase. With regards to **Q1**, we conclude that with respect to bandwidth requirements, *Origo* clearly outperforms 2PC oracles in any type of network except for the post-record phase. This means that for application scenarios, where ZKPs prove small data sizes but record transcripts are large, *Origo* is the most beneficial solution. If ZKPs prove large record sizes, then *Origo* should not be considered as a solution (cf. Figure 4.10), because *Origo* scales in the same way as existing approaches with zkSNARK proof systems.

Q2: Does replacing the 3PHS with a ZKP key derivation yield a quantifiable performance advantage in the end-to-end protocol evaluation?

Answer: Replacing the 3PHS and the 2PC-based key derivation with a ZKP circuit to derive the TLS session key removes any secure 2PC evaluations in the handshake phase (cf. Table 4.3). The comparison of ZKP circuits between *Deco*, *DecoProxy*, and *Origo* (cf. Figure 4.16) shows that the key derivation overhead remains negligible when using any proof systems. Our answer to **Q2** states that 2PC-free proxy oracles achieve efficient TLS handshake negotiations without compromising much performance in the record phase.

Q3: Based on the efficiency and security assessment, which deployment setting is ideal for the benefits provided by *Origo*?

Answer: In lossy or unreliable networks, we suggest considering the ZKP-based oracle of *Origo* due to the communication efficiency it provides (cf. Table 4.3). Another aspect to highlight is that *Origo* with the proxy mode integrates well in the web context as browsers allow configuring proxies out of the box. Further, *Origo* allows decoupling the collection of the transcript commitment and the ZKP verification. If users switch to another browser session with the Oracle verifier for the ZKP verification, then servers cannot enforce CORS policies for Oracle censorship purposes.

4.4. Policy-driven Automation of PET Computations

Modern privacy-enhancing technologies reach new forms of computation and data privacy according to a statement (e.g. private value > 100). However, beyond the statement expression, the description of a private computation circuit requires knowledge of security algorithms protecting private values.

To keep the description of private and compliant computation circuits as close to the statement expression as possible, we introduce a new composable policy language called zkPolicy. Further, we introduce a policy transpiler, called zkGen, to decouple the complexity expressed via zkPolicy from the complexity of the underlying security algorithms. Our results show that with zkPolicy, the description of compliant data provenance circuits can be reduced from 957 to 22 lines of code. And, with zkGen, we automate the generation and composition of private computation circuits to a minimum effort of configuring a zkPolicy.

Motivation: Through PETs such as ZKP systems, data, and computational privacy achieve new forms of compliance, where compliance holds against a public statement. The statement typically expresses single or multiple relations that hold on arguments that are kept private during the privacy-preserving computation. Even though statements can be as simple as a comparison of two variables (e.g. bank balance > 10.000$), the security protocols keeping variables private might rely on complex cryptographic suites and many algorithms. For instance, to prove the data provenance of web traffic, current ZKP circuits prove unambiguous mappings between API values and parameters of a TLS session and, with that, reach constraints in the range of millions [56]–[58]. Thus, creators of privacy-preserving computation circuits need domain-specific knowledge to link security algorithms to private variables and cannot solely specify circuits via a public statement.

Challenge: With upcoming proofs of web interactions [144], configurable network policies [57], and verifiable credential applications [145], not only the efficiency requirements of PETs grow. More importantly, users become exposed to selecting or configuring public statements to express the desired interaction. As such, users implicitly become the creators of circuits. The custom selection and combination of statements introduce a new dynamic, which is currently not covered by frameworks that implement PET systems. Instead, current frameworks provide privacy-preserving subcircuits which are referred to as gadgets. Gadgets are, by default, not connected to any statements such that every circuit with compliance against a statement depends on a manual composition of gadgets to assert the statement.

Contribution: As a remedy, our last contribution[18] introduces a transpiler architecture to automatically compose statement-compliant circuits for privacy-preserving computations. Our transpiler, called *zkGen*, relies on a domain-specific policy to parse the desired public statements. For instance, to generate ZKP circuits, *zkGen* takes in a ZKP-specific policy written in a new *zkPolicy* language. With the *zkPolicy* language, we focus on a higher-level description of ZKP circuits at a statement level and rely on an abstraction layer, the gadget library, to connect statement arguments to the variables of security algorithms. Thus, taking as input a

[18]Major parts of this Section 4.4 are subject to copyright protection: © 2024 IEEE. Reprinted, with permission, from [Lauinger, Ernstberger, Steinhorst, zkGen: Policy-to-Circuit Transpiler, IEEE International Conference on Blockchain and Cryptocurrency (ICBC), May/2024] (cf. Appendix D).

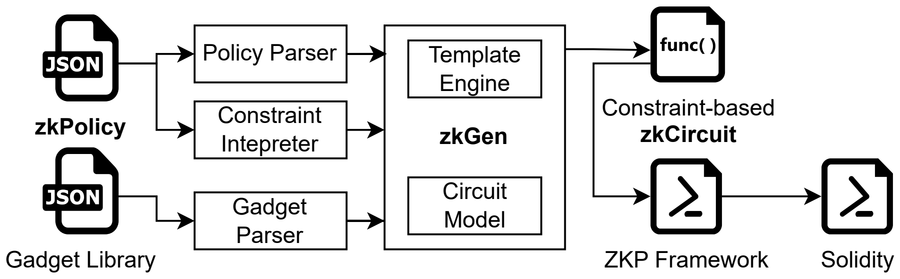


Figure 4.17.: Overview of the *Portal* software architecture which, based on a gadget library, transpiles a policy-specific circuit description (e.g. *zkPolicy*) into a constraint-based representation of a ZKP circuit. © 2024 IEEE

zkPolicy and the gadget library, *zkGen* outputs a constraint-based circuit description written in a Domain Specific Language (DSL) (cf. Figure 4.17). With that, any general-purpose ZKP framework supporting the DSL is able to process the constraint-based circuit and, if applicable, generate blockchain verification code (e.g. Solidity code for the Ethereum Virtual Machine (EVM)).

Hence, with *zkGen*, the automated generation of compliant private computations takes the next step towards a simple description with fewer lines of code than constraint-based circuit descriptions. Further, circuits with outdated or insecure gadgets can be easily reconfigured by updating a single keyword in a *zkPolicy*. In summary,

- We propose a policy language called *zkPolicy*, which expresses ZKP circuits via statements and gadget identifiers (cf. Section 4.4.1).
- We introduce a transpiler architecture to automatically generate and compose ZKP circuits according to a policy statement defined by the *zkPolicy* language (cf. Section 4.4.2).
- We open source[19] and evaluate *zkGen* and our gadget library (cf. Section 4.4.3).

4.4.1. zkPolicy Language

The *zkPolicy* language gives the opportunity to describe a ZKP circuit by connecting policy-relevant variables in a new data model. The data model builds upon a *gadget library* abstraction layer which groups ZKP subcircuits according to security algorithms.

Data Model

The *zkPolicy* language allows configurations based on the following data model, where

Relations are objects connecting two *arguments*.

[19]https://github.com/jplaui/zkGen

```
{"name": "zkPolicy_tls13openSessionCommit",
 "relations":[{
  "value": {
   "type": "s-string",
   "size": 5,
   "protection": {
    "algorithm": "secure_channels:openRecord_TLS13AES128GCMSHA256",
    "parameter": "value"
   }
  },
  "number": {"type": "p-integer"}
 }],
 "constraints": [
  "0:value->-0:number",
  "0:value:protection:algorithm:key-=-commitments:mimc:message"
 ]
}
```

Figure 4.18.: JSON file expressing a ZKP circuit with the zkPolicy language. © 2024 IEEE

Arguments are key-value pairs, where keys express the name of an *argument*. Argument keys are unique in the scope of a relation. Argument values are objects with three key-value pairs; a *type*, a *size*, and a *protection*.

Types specify if the *argument* counts as secret or public by concatenating a "s-" or "p-" with an argument type t. Currently, *zkGen* supports two argument types with $t \in \{\text{string}, \text{integer}\}$.

Sizes specify the number of characters in an *argument* of type t=string and do not exist for *arguments* of type t=integer.

Protections use objects as values and rely on two key-value pairs to connect an *argument* to a security algorithm. To do so, *protections* point to a security algorithm of the *gadget library* via a string identifier "algorithm_type:algorithm_id" (cf. Figure 4.18). Further, *protections* map an *argument* to a security algorithm parameter via the "parameter" key.

Constraints are strings expressing assertions between elements such as *arguments*, *protections*, or both. Assertions rely on comparison operators (e.g. $<,>,\in$, etc) between two dashes. Multiple dashes express a logical OR between appended elements and the first element. Multiple constraints on the same element express a logical AND. The key words *if* and *for* follow a transpiler-specific syntax and are used to express conditions and loops. If *constraints* reference *protection* parameters, the *protection* string identifier is combined with the parameter name. If *constraints* reference *arguments*, the *relation* index is combined with the argument key or argument path towards a protection parameter.

zkObjects represent a complete *zkGen* policy configuration using the *zkPolicy* language (cf. Figure 4.18). The *zkObject* comprises three key-value pairs which define a list of *relations*, a list of *constraints*, and a name.

```
{"commitments": {
  "mimc": {
    "commit_string":
      {"type": "p-string", "size": 32},
    "witness":
      {"type": "s-string", "size": -1},
    "message":
      {"type": "s-string", "size": -1}
}}}
```

Figure 4.19.: Simplified entry of the *gadget library* that defines the parameters of an mimc hash computation. The algorithm_type is set to cryptographic commitments and the algorithm_id is the keyword "mimc". Parameters of arbitrary size are indicated with a size=-1. © 2024 IEEE

Gadget Library

The *gadget library* abstracts security algorithms to public and private parameters and uniquely defines algorithmic abstractions by grouping identifying names per algorithm under algorithm types (cf. Figure 4.19). Notice that the *zkGen* transpiler requires an implementation of every gadget in the DSL of the respective PET system. The *gadget library* currently supports different abstractions, where

Cryptographic commitments depend on a commitment string as the public parameter and a witness and a message as the private parameters (cf. Figure 4.19).

Encryption algorithms currently exist in two forms, where symmetric, asymmetric, and the one-time-pad encryption algorithms depend on a plaintext and a key as the private parameters and a ciphertext as the public parameter. *Encryption algorithms* in the counter mode require additional public parameters with an initialization vector and a block index.

Digital signatures are represented via two private parameters with the message and the signature value itself and a public key as the public parameter. In the context of anonymous credential systems [54], [55], proving knowledge of signatures is combined with a statement evaluation of the signed data.

Secure channels are constructed by the composition of an encryption gadget and a single or multiple commitment gadgets. In the context of data provenance proofs [12], [56], [58], ZKP circuits allow users to validate web traffic secured by TLS against a statement (e.g. bank balance > 100$).

Compositions

The *zkPolicy* together with the *gadget library* allows the composition of multiple relations in a single policy, where relations are linked together by constraints. For example, credential chaining as introduced in the work [55] can be achieved by using two relations r_1, r_2 with arguments that refer to the respective credential values via the constraints:

1. "0:age->-0:number"
2. "1:age-<-1:number"
3. "0:age:p:a:commit_string-=-1:age:p:a:commit_string"
4. "0:age:p:a:witness-=-1:age:p:a:witness"

Here, the characters "p:a" shorten the "protection" and "algorithm" path keywords of the *zkPolicy* language and the indices 0,1 refer to the relation indices. Credential chaining is used to evaluate multiple credentials in a single proof verification.

4.4.2. zkGen Transpiler

The following subsections introduce the components and capabilities of the *zkGen* transpiler as a command line toolkit.

Gadget Parser

Before a policy is parsed, the transpiler reads in the *gadget library* in the form of a *library.json* file. In the next stage, the transpiler iterates over all security algorithms of the *gadget library* and searches for every gadget implementation. To successfully parse a gadget implementation, gadgets must be written under certain rules. Independent of the DSL a gadget is written in, *zkGen* relies on specific comments to facilitate the gadget parsing. Since gadget implementations can grow into many files and folders, a typical comment location is before the beginning and end of a circuit definition. If the gadget exists as a module, then comments must indicate the module name to enable seamless imports of other gadgets.

Policy Parser

The policy parser reads in policies as JavaScript Object Notation (JSON) files and iterates over the *relation* arguments to identify a list of *protection* algorithms per *argument*. The sequence of *protection* algorithms is arranged by the constraint interpreter and used to identify gadget implementations for the composition of a circuit model. If an *argument* is of type string, the policy parser extends the list of *protection* algorithms at an *argument* with an extra gadget that converts the input argument to an aggregated integer representation. We present the string-to-integer conversion logic in the Formula 4.2, where the function $\texttt{ascii}(\text{str}_i)$ returns the American Standard Code for Information Interchange (ASCII) number of a string character and $len(\text{str})$ indicates the length of the string str.

$$a^{\text{aggr}} = \sum_{i=0}^{len(\text{str})} 10^i \cdot (48 - \texttt{ascii}(\text{str}_{len(\text{str})-i})) \tag{4.2}$$

In the next stage, the parser module reads in and shares the list of *constraints* with the constraint interpreter module.

Constraint Interpreter

The constraint interpreter processes the list of *protection* algorithms and *constraints* and creates data attributes for the template engine. By iterating over the lists of the policy parser, the constraint interpreter removes duplicate references of the same security algorithm such that the circuit template instantiates every required security algorithm once. Any constraint duplicates per argument are removed as well. Next, the constraint interpreter builds a list of data attributes to encode the string-to-integer conversion per input argument. The second list of data attributes is used to instantiate and sequentially apply *protection* algorithms to arguments. The last list of data attributes encodes constraint checks between public and private arguments. The constraint interpreter throws errors if private arguments potentially leak information (e.g. equality check of private argument against public argument).

Circuit Models & Template Engine

The output of the transpiler is generated based on a circuit model which *zkGen* maintains via a template engine. To start the composition of the circuit model, *zkGen* uses the template engine to add all string-to-integer conversion gadgets per input argument. Next, the required *protection* algorithms per argument augment the circuit model. To add the protection algorithms to the circuit model, *zkGen* relies on the parsed gadget implementations and the second list of data attributes of the constraint interpreter. If a security algorithm is called multiple times, then the circuit model resets all initialization variables of the algorithm. Last, the template engine adds individual constraints per argument to the circuit model.

In the end, *zkGen* generates circuits by executing the template engine on a circuit model. The outputs of the template engine are strings which are stored in unique output folders. Before a string is stored, *zkGen* determines the filename extension of output files based on the DSL the circuit is written in. The *zkGen* transpiler supports the generation of testing suites. However, to generate a test, the values of private and public arguments must be provided in a test configuration. The execution of a generated test suite depends on the installation of the desired ZKP system. If the ZKP system is installed, *zkGen* supports the generation of blockchain code (e.g. Solidity) that verifies the generated circuit.

4.4.3. Evaluation & Discussion

The evaluation summarizes the results and compares *zkGen* against related works.

Table 4.5.: zkGen transpiler benchmarks. © 2024 IEEE

zkPolicy	LOC policy	LOC circuit	Transpile Time
TLS 1.3 commit	22	975	2.38 ms
Age commit	20	85	1.70 ms

Transpilation Results

Our implementation[20] of *zkGen* relies on the *text/template* package as the template engine and uses the *gnark* ZKP framework [98] to (i) generate the ZKP verification code in Solidity and to (ii) compute, verify, and test ZKPs. The *zkPolicy* language reduces the description complexity of a policy-compliant ZKP circuit with regard to Lines of Code (LOC). Further, our toolkit achieves transpile times in the range of milliseconds (cf. Table 4.5).

Related Works

The work *zkDocs* [145] introduces a transpiler toolkit to generate customizable privacy documents based on readable JSON schemas. Schemas of *zkDocs* let users define lists of fields, constraints, and trusted institutions, where fields are protected by commitments and asserted against defined constraints. Every constraint follows the syntax

$$<f_A><\text{SUB/ADD}><f_B><\text{LT/GT}><\text{const}/f_{Comp}> \tag{4.3}$$

with fields f, the subtract and addition operators as SUB, ADD, and the comparison constraints larger than or greater then as LT, GT. The list of trusted institutions is used as an access control list to protect attestations on commitments. The generated circuit computes a fixed commitment check for each schema field and adds constraints, as additional assertions, on fields. In contrast to *zkDocs*, the *zkGen* can generate circuits with multiple distinct security algorithms and introduces a flexible *zkPolicy* language with enhanced expressiveness.

The work *DataCapsule* [146] introduces a flexible policy language called *PrivPolicy* with complete expressiveness. *PrivPolicy* extends the *Legalease* policy language [147] by using residual policies to describe the minimum and maximum privacy restrictions of processed data. With residual policies, *PrivPolicy* is applicable to programs that protect information leakage of private data processing through the concept of differential privacy [148]. In contrast to *zkGen*, *DataCapsule* statically analyzes programs to guarantee policy-compliant data processing but does not generate policy-compliant programs. However, since ZKP circuits do not support differential privacy [89], we deem *DataCapsule* as an interesting related work to investigate if differential privacy compliance could be added to the generation of ZKP circuits.

[20]https://github.com/jplaui/zkGen

Limitations & Future Work

The first limitation of the *zkGen* toolkit is that it supports the generation of ZKP circuits only. With MPC or HE as other PETs, we deem the generation of compliant MPC or HE circuits as future work. Secondly, as the *zkPolicy* language lacks a formal definition and is limited to a specific PET domain, we see the formal definition of the *zkPolicy* language and the definition of a domain-independent PET policy language as future work. The future goal of formalizing the *zkPolicy* language is to attribute generated PET circuits with sound privacy guarantees. Integrating tools to formally verify generated circuits is another direction for future work.

5. Final Discussion

Our final discussion delivers answers to our initially defined research questions of Chapter 1. Based on the state of progress achieved in this work, we summarize interesting directions of future research and provide an outlook on how our proof of concepts and contributions help the research community and industry. Then, we end our work with concluding remarks.

5.1. Answering Research Questions

In the following, we restate our research questions and answer how and to which degree our contributions solve the research questions.

Research Question 1: How can we ensure self-determined protection and transparency of digital data during the creation and interaction of online accounts?

As an important contribution, our first systematization [84] defines the level and notion of data sovereignty that can be built with today's cryptography. As such, we provide a conceptual definition of self-determined protection and transparency of digital data. At this point, it remains unclear how the concepts apply to existing online infrastructures and deployments.

In our work *Portal* [82], we construct a time-bound and replay-resistant ZKP verification at smart contracts. On top, we present a modern identity system with enhanced privacy and control. *Portal* respects our initially defined principles of data sovereignty and data privacy in the context of identifier and data administration. In *Portal*, *users* have the responsibility and control to manage, verify, and present data through the public blockchain while preserving data privacy. If *Portal users* present data at the *Portal* plugin, then the verifier can choose between inspecting past validations of *user* data or requesting a live verification of user data. During the *Portal* SSO, *users* are less susceptible to potential tracking compared to traditional SSO solutions. If *users* switch wallets for every live verification, *users* cannot be tracked at all. *Portal* serves as the first SSO alternative with conventional usability that gives *users* a choice to pick enhanced control and privacy during authentication at small costs.

With A-PoA [83], privacy-preserving certification ecosystems gain the ability to anonymously authorize parties to issue credentials according to a credential schema. Our construction uses the RSA-accumulator to authorize credential issuers and relies on the privacy-preserving NI-ZKPoKE protocol to prove the accumulator membership. Our solution enables resource-efficient and privacy-preserving membership authentication with little communication overhead. Based on our work, privacy-preserving, secure, and verifiable hierarchies or relationships between authorities can be added to the sovereign, VC-compatible ecosystem.

To summarize and address our first research question, our contributions define the theory and practice to bootstrap a system with self-determined protection of digital data. Transparency is balanced through the use of advanced ZKP technology. *Portal* accounts achieve a strong notion of data sovereignty, are cost-efficient, and accountable. Our work A-PoA enhances privacy and security for a verifiable but private creation of trustworthy online accounts that issue attestations. Both contributions remain compatible with the W3C's standardization activities for sovereign applications in a DID and VC-based ecosystem. All things considered, our first contributions show how a system of sovereign accounts maximizes data privacy during online interactions.

Research Question 2: How can we practically return the sovereignty of custodial data back to devices and users?

With the second systematization, we investigate state-of-the-art approaches to verify the provenance of data. We perform a complexity analysis of TLS oracles on a function level to identify major limitations that influence the practicality of existing approaches. We identify that for MITM-secure oracles, the secure computation overhead concerning bandwidth requirements impacts the deployment of TLS oracles in constrained environments. Further, we notice that the efficiency of proof computations during the challenge phase yields performance benchmarks in the range of minutes, which impacts a practical user experience of the TLS oracle client side.

To address the identified bottlenecks, our work *Janus* [58] reconsiders the selection of secure computation techniques in TLS oracles by putting an emphasis on the asymmetric privacy setting and the conditions found in TLS 1.3. Concerning the asymmetric privacy setting, we show that a HVZK proof system can be deployed if the proving party performs a unilateral validation check of the *verifier*. Concerning TLS 1.3 in the 1-RTT mode, we show that the authenticity of SHTS can lower algorithmic security requirements in the garble-then-prove paradigm. The outcomes of *Janus* improve the efficiency of ZKP computations and improve E2E benchmarks.

In our work *Origo* [97], we introduce a new type of oracle that does not rely on secure 2PC. We show that *Origo* achieves the best performance benchmarks in the TLS handshake phase while adding a negligible overhead in the TLS record phase. *Origo* lacks support for verifying larger data sizes which the *Janus* contribution solves. Thus, we conclude that by merging our findings, TLS oracles become verify efficient with respect to all TLS phases.

To summarize and answer our second research question, we find that a tailored interplay between security assumptions, adapted PET technologies, and protocol-specific conditions leads to substantial improvements for protocols that reclaim ownership of digital data. Our results significantly enhance state-of-the-art approaches and advance the sovereignty over shared kilobyte-sized digital assets. Our insight that, in the asymmetric privacy setting, proof systems can be reduced to a HVZK setting with malicious verifiers is of independent theoretical interest to the research community. Now, researchers can investigate if and how protocols can be transferred into an asymmetric privacy setting in order to benefit from efficient HVZK proofs.

Research Question 3: How can we mitigate the unconscious sharing and processing of custodial data through user-driven policies?

Via our systematization of data sovereignty, we notice that preventing the unconscious sharing and processing of data is only possible if the data remains protected at all times. However, the level of protection remains configurable and must remain in the hands of the device or the user. The extent of data protection is defined by PET technologies and we conclude that access to policy-driven PET technologies becomes inevitable.

As such, with our contribution *zkGen*, we automate the policy-compliant generation of computational privacy via a transpiler architecture. Instead of manually configuring computational assertions with regard to a policy, we introduce a new policy language to describe and connect private computation variables to policy statements. The policy language delivers a concise description of compliant and private computation circuits with a fraction of code lines compared to constraint-based circuit descriptions. Together with a data sovereign system (e.g. our contribution *Portal*) or combined with sovereign protocols to reclaim digital ownership (e.g. our contributions *Janus*), *zkGen* enables a policy-driven level of data sovereignty.

To answer our third research question, we can say that the unconscious sharing and processing of digital data is mitigated if devices control every operation on data through policy-driven technologies. It must hold that custodial data operations remain verifiable such that a policy-governed execution can be guaranteed. An example deployment of policy-driven applications can be achieved via an accountable system (e.g. *Portal*), where every execution or sharing of data yields a transaction that can be checked against the policy.

5.2. Outlook & Future Work

This thesis proposes solutions for an independent and sovereign management of data and identifiers. Additionally, this work shows how devices reclaim ownership of shared data which resides at centralized services. At this stage, our work does not investigate how data can be shared and processed responsibly. We neither define what constitutes mediating practices with sufficient accountability and privacy. Hence, this outlook discusses emerging challenges of mediated but sovereign data sharing and processing. At the same time, we summarize future work ideas that could enhance the verification of content provenance even further.

Sovereignty for Mediated Data Administration

The challenge of mediated sovereignty is the enforcement of a verifiable and accountable transaction for every mediated action. For example, concerning data sharing, a device must be able to resolve the access privilege and time when a data consumer obtains access to data.

For data with stronger protection requirements, a device must be able to identify, audit, and, under certain circumstances, punish the computing party as well as the data consumer.

Based on our systematization of data sovereignty, a challenge of the future is the efficient integration of secret management committees into decentralized public infrastructures [84], [88]. The functionality of secret management in combination with a transaction-based state replication enables the principle of *extractable witness encryption* [149]–[151], where a message is encrypted under a problem instance which the decryptor must know in order to access the message. However, proactive secret management in dynamically changing networks has been investigated but remains challenging to deploy at a larger scale [152].

In order to achieve policy-driven, private, and verifiable computation, the model of extractable witness encryption must be extended as follows. Two parties apply for the witness extractor, where each party has a different task. The first party, the compute node, must access the data, perform a computation, and attest to the policy-compliant execution of a function. The data consumer must be able to fetch the result from the compute node once the execution is finalized. The abstraction of policy-driven computation is currently not covered by the model of extractable witness encryption and poses an interesting direction for theoretical future research.

Data Provenance for Production Systems

In the field of data provenance protocols, we identify two main problems as promising future work directions. The first problem concerns a feasibility analysis of commitment attacks in the context of data provenance protocols. Commitment attacks, such as the message franking attack, allow an adversary to find a second key message pair that encrypts to the same ciphertext and authentication tag as another key message pair [126]. When performing the attack, the adversary typically finds a second message composed of garbage characters. However, in TLS oracles, the attacker must be able to determine one or more characters in order to generate a convincing proof for a statement. For instance, if the adversary wants to obtain an TLS data attestation on a false claim (e.g. bank balance > 10.000), then the adversary must be capable of generating a sequence of at least 5 digits in the franked message. An analysis of the feasibility of such an attack has not been done yet. If such an attack is difficult, then the *Origo* and *Janus* TLS oracles could be further optimized by removing the key derivation parts of the ZKP circuits.

Another topic of future research is the censorship resistance of TLS oracles. Even though current approaches claim to achieve strong censorship resistance (e.g. TLS oracles in the notary mode [11], [12]), server-side CORS policies could restrict clients from fetching external resources. At the same time, the proxy mode is susceptible to censorship based on IP address analysis. Finding strong guarantees for censorship resistance is a topic that might compromise the technical applicability of data provenance protocols in the future.

5.3. Concluding Remarks

In response to continuing activities of digital legislation on data protection, we investigate and advance technologies that empower a sovereign and privacy-preserving administration of digital data.

To do so, this work systematizes the notion of digital sovereignty which can be constructed today. Based on our insights, we propose two important concepts for self-determined identity and privacy-preserving management of credentials, which comply with the current standardization activities of the W3C. Additionally, this thesis systematizes approaches to reclaim control over shared digital data that resides at external storage locations. This analysis yields important insights and leads to two core contributions that improve the efficiency of verifying the provenance of data protected by secure channels. To enable the policy-driven processing of data through privacy-enhancing technologies (PETs), this work proposes a transpiler toolkit for an automated generation of secure computation building blocks. The contributions of this thesis can be used in conjunction with each other in order to amplify the notion of sovereignty in the system. We provide practical implementations and open-source building blocks to the public, academia, the research community, and the industry.

In summary, this work proposes sovereign online functionalities that serve as independent components to build privacy-preserving and data-sovereign applications. The concepts and findings identified in this work remain applicable even if PETs continue to improve in the future.

A. Security Analysis

This chapter groups the security proofs of the works *Portal* [82], *A-PoA* [83], and *Janus* [58].

A.1. Portal Security Proof

The security analysis shows that the time-bound ZKP verification at smart contracts is resistant to replay attacks under the following threat model:

Theorem 1. *If a party p_1 with access to*

- *a smart contract $\mathcal{C}^{\mathcal{C}_1}$*
- *a secure proof system Π_π*
- *a secure signature scheme Π_σ*
- *a secure hash function Π_H*

performs the sequence of computations

1. *p_1 compiles and signs a transaction tx_1 with Π_σ.Sign*
2. *p_1 calls $\mathcal{C}^{\mathcal{C}_1}$.sample with tx_1 such that $\mathcal{C}^{\mathcal{C}_1}$ generates the prevrandao randomness r using Π_H.Hash and stores $m[W^{p_1}_{addr}]r||''\text{-}0''$ at timestamp t_1*
3. *p_1 fetches r from $m[W^{p_1}_{addr}]$*
4. *p_1 computes π=Π_π.Prove($ccs_{\mathcal{C}_1}$,w,pk)*
5. *p_1 compiles and signs a transaction tx_2 with Π_σ.Sign, where $\pi \in tx_2.d_{pl}$*
6. *p_1 calls $\mathcal{C}^{\mathcal{C}_1}$.verify with tx_2 and $\mathcal{C}^{\mathcal{C}_1}$ sets $m[W^{p_1}_{addr}]r||''\text{-}1''$ at timestamp $t_2 > t_1$*

under the assumptions that

- *$\mathcal{C}^{\mathcal{C}_1}$ runs on a blockchain which guarantees liveness, consistency, safety*

we say that the proof π is resistant against replay attacks performed by a malicious PPT adversary such that $\pi \in tx_2{'}$ is never accepted by $\mathcal{C}^{\mathcal{C}_1}$. And, we say that computing π is bound by the time t_1 and cannot be accepted after t_2.

Proof 1. At the time t_1, the adversary $\mathcal{A}$ cannot change tx_1 of p_1 as unforgeability of transaction signatures holds. But, $\mathcal{A}$ is capable of registering the same nonce of p_1 twice at $\mathcal{C}^{\mathcal{C}_1}$ with $\text{tx}_1{}'$. $\mathcal{C}^{\mathcal{C}_1}$ maps the nonce of $\mathcal{A}$ at the address $m[W^{\mathcal{A}}_{addr}]$. At time $t > t_1$, $\mathcal{A}$ uses the blockchain logs to access tx_2 of p_1 and, with that, π. If $\mathcal{A}$ replays π in a transaction $\text{tx}_2{}'$ and calls $\mathcal{C}^{\mathcal{C}_1}$.verify, then the verification of circuit $\mathcal{C}_1$ fails because of the following issue. The proof π has been computed with the address $m[W_{addr}]$ as private input and $\mathcal{C}^{\mathcal{C}_1}$ asserts that π is verified against the owner of tx_2. $\mathcal{A}$ has signed $tx_2{}'$ such that $\mathcal{C}^{\mathcal{C}_1}$ asserts π with the public input $W^{\mathcal{A}}_{addr}$ taken from $tx_2{}'$, which fails.

Further, $\mathcal{A}$ tries to replay a previously accepted proof $\pi^{\mathcal{A}}$ (sampled and proven with $tx^{\mathcal{A}}_1$ and $tx^{\mathcal{A}}_2$). At the time $t > t_1, t_2$, $\mathcal{A}$ cannot replay $\pi^{\mathcal{A}}$ because, even though a reoccurring nonce is negligible (collision resistance of PoS randomness), $\mathcal{C}^{\mathcal{C}_1}$ prevents overwriting an existing map entry at $m[W^{\mathcal{A}}_{addr}]$ if a nonce has already been set. Thus, our scheme is *replay-resistant* and *time-bound* as π can only be computed at t_2 after randomness has been sampled with tx_1 at time $t_1 < t_2$.

A.2. A-PoA Security Proofs

The security analysis focuses on protocol integrity and accumulator element confidentiality. The analysis of all parameters references the definitions under use and proves the respective security assumption that applies to the parameters.

Theorem 2. *Suppose authenticated encryption of messages throughout the communication protocols holds, then the authentication scheme introduced above is secure against $\mathcal{A}_1$ intending to eavesdrop and modify or decrypt messages m.*

Proof Supposing an adversary $\mathcal{B}$ that is able to provide $\mathcal{A}_1$ with the private keys (through guessing or collision) associated with the DIDs used during message exchange contradicts the security assumptions of authenticated encryption. Keys used in authenticated encryption are based on asymmetric cryptography which relies on e.g. the strong RSA assumption (introduced below).

Definition 1. *(Strong RSA assumption in generic groups of unknown order [45]) There is no probabilistic polynomial-time algorithm $\mathcal{P}$ that outputs w and an odd prime x such that $g^x \equiv u \pmod{n}$, except with negligible probability:*

$$PR\left[w^x \equiv g : \begin{array}{l} \mathbb{G}_? \xleftarrow{R} Gen(\lambda),\ g \xleftarrow{R} \mathbb{G}_? \\ w, x \in \mathbb{G}_? \times \mathbb{Z} \leftarrow \mathcal{A}_2(\mathbb{G}_?, g) \end{array} \right] < negl(\lambda)$$

Theorem 3. *Under the strong RSA assumption, the above-mentioned RSA accumulator used in our scheme is secure against $\mathcal{A}_2$ and forging of membership.*

Proof In order to prove security against adversary $\mathcal{A}_2$, suppose there exists an adversary $\mathcal{B}$ that is able to find a collision, hence obtaining $\{x_1, x_2, ..., x_n, x', a'\}$ such that $(a')^{x'} = g^{x_1,...,x_n} \pmod{n}$. $\mathcal{A}_2$ is given access to all public values initiated in the generation phase.

Leveraging $\mathcal{B}$, $\mathcal{A}_2$ can break the RSA assumption as described in [43]. Let $x = x'$ and $r = x_1, ..., x_n$. Compute $\alpha, \beta \in \mathbb{Z}$ as Bezout coefficients for $\alpha \cdot r + \beta \cdot x' = 1$. Set $w = (a')^{\alpha} g^{\beta}$ satisfying $w^x \equiv g$. This indicates that if a value x_i is once revoked from the accumulator, the entity authorized before is unable to efficiently obtain membership without authentication.

Theorem 4. *Our scheme is secure against $\mathcal{A}_3$ if the one-time interaction between P and V can be simulated through an efficient simulator $\mathcal{S}$ and the Fiat-Shamir heuristic applies.*

Proof (Sketch). In order to prove security against $\mathcal{A}_3$, we rely on the simulator introduced by Boneh et al. in [45]. It is argued that the *GenProof* function can be efficiently simulated by $\mathcal{S}$, outputting $\{z', Q'_g, Q'_{w_x}, r'_x, r'_\rho\}$ which is statistically indistinguishable from the real protocol/transcript execution $\{z, Q_g, Q_{w_x}, r_x, r_\rho\}$. Hence, transcript T_{sim} generated through $\mathcal{S}$ is indistinguishable from T_{real}, satisfying the HVZK property for a proof of knowledge as described in Sec. C.1. Note that the NI-ZKPoKE is general zero-knowledge in the random oracle through the Fiat-Shamir heuristic. The reason for this is the replacement of the challenge with a hash which prevents $\mathcal{A}_3$ from guessing the challenge in advance.

A.3. Janus Security Proofs

The security analysis concerns the deployment of the HVZK proof system and the unilateral validation in the asymmetric privacy setting. Further, we show that the *Janus* protocol is secure against malicious adversaries during the mutual authentication of the SHTS parameter. The security analysis relies on our threat and system model (cf. Section 4.2.1) and uses our formalized cryptographic building blocks (cf. Sections 2.2.2).

A.3.1. Construction 1

The first construction creates a maliciously secure evaluation of the HVZK proof system in the asymmetric privacy setting. The proof system leverages semi-honest 2PC based on boolean garbled circuit [70] and is combined with a unilateral validation phase. To show the security of the construction, we first define the security guarantees of the asymmetric privacy setting and conclude that the unilateral validation protocol patches remaining vulnerabilities.

Theorem 5. *If three parties p_0, p_1, and p_2 with access to*

- *a three-party TLS handshake protocol Π_{3PHS}*
- *a secure commitment scheme Π_{com}*
- *a secret sharing scheme Π_{ss} with p_0 as the trusted dealer*
- *a secure channel $sc_{0\text{-}1}$ between p_0 and p_1*
- *a secure channel $sc_{1\text{-}2}$ between p_1 and p_2*
- *a secure channel $sc_{0\text{-}2}$ between p_0 and p_2*

- *a maliciously secure* 2PC *scheme* Π_{2PC} *between* p_1 *and* p_2

perform the sequence of computations

1. p_0 *calls* [r_1, r_2]= Π_{ss}.*Share*(r), *with* $r \xleftarrow{\$} \mathcal{R}(\lambda)$
2. p_0 *shares* r_1 *using* sc_{0-1} *and* r_2 *using* sc_{0-2}
3. *either* p_0 *calls* c=Π_{com}.*Commit*(m, r) *with bit strings* m, c *and shares* m, c *using* sc_{0-1} *and* c *using* sc_{0-2}, *or*

   Π_{2PC} *evaluates* Π_{com}.*Commit*(m, r_1+r_2) *where* p_1 *has* m
4. p_2 *shares* r_2 *using* sc_{1-2}

under the assumptions that

- *the TLS 3PHS implements* Π_{ss} *and the sequence of computations (1) and (2)*
- p_0 *discards calling* Π_{com}.*Open*
- p_0 *cannot be compromised by the adversary*
- p_1 *never discloses the secret share* r_1
- *the security of the schemes* Π_{2PC}, Π_{3PHS}, *etc. holds (e.g. 3PHS relies on the discrete logarithm hardness to find a from aG, with random* $a \xleftarrow{\$} EC(\mathbb{F}_p)$ *and base point* $G \in EC(\mathbb{F}_p)$*)*

we say that asymmetric privacy holds between p_1 *and* p_2 *such that only* p_1 *can call* Π_{com}.*Open.*

Proof 1.1: The security of the 3PHS keeps secret shares confidential. Without access to the initially shared secret shares, the adversary $\mathcal{A}$ cannot compute the commitment string c. Further, the security of the commitment scheme prevents the adversary from finding a collision of c. When computing the commitment through a maliciously secure 2PC system, then $\mathcal{A}$ cannot learn any information on the inputs of the counterparty. Since all parties use secure channels to communicate parameters, $\mathcal{A}$ learns nothing of communicated parameters. Thus, $\mathcal{A}$ cannot find any m or reconstruct r which prevents $\mathcal{A}$ from calling Π_{com}.Open.

Theorem 6. *If two parties* p_1 *and* p_2 *with access to*

- *a HVZK proof system* Π_{HVZK} *using a semi-honest* 2PC *system* Π_{sh2PC}
- *two secure commitment scheme* Π^1_{com}, Π^2_{com}
- *an asymmetric privacy setting* Π_{asym} *using* Π^2_{com}
- *a* 2PC *circuit* $\mathcal{C}_{open}$ *implementing* Π^2_{com}.*Open*
- *a secure channel* $sc_{1\text{-}2}$ *between* p_1 *and* p_2
- *a unilateral validation* Π_{uv} *using* Π^2_{com}

perform the sequence of computations

1. Π_{HVZK}*.Setup:* p_2 *calls p=*Π_{sh2PC}*.Garble(*$\mathcal{C}_{open}$*)*
2. Π_{HVZK}*.Setup:* p_2 *shares* $\{p \setminus T_{k-d}\}$ *using* $sc_{1\text{-}2}$
3. Π_{HVZK}*.Prove:* p_1 *calls k=*Π_{sh2PC}*.Evaluate*
4. Π_{uv}*:* p_1 *calls c=*Π^1_{com}*.Commit(k,r) with* $r \xleftarrow{\$} \mathcal{R}(\lambda)$
5. Π_{uv}*:* p_1 *shares c using* $sc_{1\text{-}2}$
6. Π_{uv}*:* p_2 *shares* $\{p\}$ *using* $sc_{1\text{-}2}$
7. Π_{uv}*:* p_1 *recomputes* $\mathcal{C}_{open}$ *to verify* $\{p\}$
8. Π_{uv}*:* p_1 *shares r using* $sc_{1\text{-}2}$
9. Π_{HVZK}*.Verify:* p_2 *calls* Π^1_{com}*.Open(c,r)*

under the assumptions that

- *in* Π_{sh2PC} p_1 *acts as the evaluator and* p_2 *acts as the garbler*
- Π_{asym} *gives* p_1 *access to* Π^2_{com}*.Commit*

we say that after running Π_{asym}*, composition of* Π_{HVZK} *and* Π_{uv} *as* Π_{comp} *establishes security against malicious adversaries.*

Proof 1.2: The security of Π_{sh2PC} allows the adversary $\mathcal{A}$ to maliciously garble the circuit $\mathcal{C}_{\text{open}}$. However, if $\mathcal{A}$ receives c upon disclosure of $\{\text{p} \setminus T_{k-d}\}$, the hiding property of Π_{com} prevents $\mathcal{A}$ from learning any secret information on the 2PC inputs of p_1. Further, p_1 detects a cheating $\mathcal{A}$ at the sequence number (7) and aborts the protocol before disclosing r to $\mathcal{A}$. Further, Π_{sh2PC} prevents $\mathcal{A}$ from predicting a k that corresponds to a 1. If $\mathcal{A}$ uses Π^1_{com} to commit garbage, then p_2 aborts at the sequence number (9).

Notice. We define Π_{comp}(Π_{sh2PC}=arg1, $\mathcal{C}_{\text{open}}$=arg2, Π^2_{com}=arg3) as an construction that takes as input a semi-honest 2PC system which is executed in the context of the HVZK proof system. The HVZK proof system evaluates a 2PC circuit as the second argument. The third argument is a commitment scheme that establishes the asymmetric privacy setting.

A.3.2. Construction 2

The second construction provides the *verifier* with a secure authenticity verification of the TLS 1.3 SHTS secret in a setting with malicious adversaries. To do so, the construction combines the effects of the TLS 1.3 1-RTT mode with the TLS 3PHS and a secure 2PC computation of the session secret SHTS. This combination introduces an unsolvable challenge to the adversary which prevents the adversary from forging the authenticity of SHTS.

Theorem 7. *If three parties* p_0*,* p_1*, and* p_2 *with access to*

- *a secure channel* $sc_{0\text{-}1}$ *between* p_0 *and* p_1
- *a secure channel* $sc_{0\text{-}2}$ *between* p_0 *and* p_2
- *a three-party TLS handshake protocol* Π_{3PHS}
- *a secure commitment scheme* Π_{com}
- *a maliciously secure* 2PC *scheme* Π_{2PC} *between* p_1 *and* p_2
- *a secret sharing scheme* Π_{ss} *with* p_0 *as the trusted dealer*
- *a secure AEAD scheme* Π_{AEAD}
- *a secure signature scheme* Π_σ *where* p_0 *maintains the private key sk*

perform the sequence of computations

1. p_0 *calls [*r_1*,* r_2*]=* Π_{ss}*.Share(r), with r* $\overset{\$}{\leftarrow}$ $\mathcal{R}(\lambda)$
2. p_0 *shares* r_1 *using* sc_{0-1} *and* r_2 *using* sc_{0-2}
3. p_2 *samples t* $\overset{\$}{\leftarrow}$ $\mathcal{R}(\lambda)$ *and discloses t*
4. p_0 *calls c=*Π_{com}*.Commit(t, r), with bit strings c*
5. p_0 *calls* σ*=*Π_σ*.Sign(sk, t)*
6. p_0 *calls s=*Π_{AEAD}*.Seal(c,*σ*) and discloses s*
7. Π_{2PC} *evaluates* Π_{com}*.Commit(t,* r_1*+*r_2*)*
8. p_2 *calls* σ*=*Π_{AEAD}*.Open(c,s) and checks* Π_σ*.Verify(pk,t,* σ*)*

under the assumptions that

- *the TLS 3PHS implements* Π_{ss} *and the sequence of computations (1) and (2)*
- p_0 *cannot be compromised by the adversary*
- *pk, and t are public*
- p_0 *never discloses sk*
- p_2 *only performs step (7) if a s has been captured*

we say that an PPT adversary has negligible probability with respect to λ *in forging c such that* p_2 *accepts step (8) and that c is authentic.*

Proof 2.1: Again, Π_{3PHS} and Π_{2PC} keep the secret shares confidential. Thus, the adversary $\mathcal{A}$ can only access c at step (7). With c, the adversary can forge a new transcript s but cannot change a s which has already been captured by p_2. Thus, the challenge for $\mathcal{A}$ is to predict a valid c' at a point in time where c remains hidden. Predicting a correct c requires $\mathcal{A}$ either to find a collision for c which the secure commitment prevents. Or, $\mathcal{A}$ correctly guesses the secret share r_2 which evaluates to a correct c before a s is captured by p_2. In the case of a correct guess, $\mathcal{A}$ can replay a σ' on previous t' and encrypt σ' under the right c such that p_2 accepts. However, guessing r_2 or r_1 has negligible probability in λ.

A.3.3. Construction 3

The third construction reduces the security requirements of cryptographic constructions in the garble-then-prove paradigm [117]. Specifically, we show that the existence of a computation trace to an authenticated commitment string allows replacing a semi-honest 2PC system based on authenticated garbling with a semi-honest 2PC system that does not require authenticated garbling. Our garble-then-prove paradigm leverages the efficient proof system construction in the asymmetric privacy setting in the proving phase. Further, it requires commitment authenticity through SHTS. Thus, for this construction, we use our definitions of Π_{comp} and Π_{auth} (cf. proof 1.1, 1.2, and 2.1 of Appendix A.3).

Theorem 8. *If two parties p_1 and p_2 with access to*

- *a garble-then-prove scheme $\Pi_{g\text{-}t\text{-}p}$ using two semi-honest 2PC system Π^1_{sh2PC} , Π^2_{sh2PC}*
- *a composition scheme Π_{comp}*
- *a secure commitment scheme Π_{com}*
- *an authenticated commitment scheme Π_{auth} using Π_{com}*
- *a 2PC circuit $\mathcal{C}_{open}$ implementing Π_{com}.Open*
- *a 2PC circuit $\mathcal{C}_{kdc+record}$ implementing the TLS 1.3 specification*
- *a 2PC circuit $\mathcal{C}_{\phi}$ implementing a data compliance check against a statement ϕ*

perform the sequence of computations

1. *$\Pi_{g\text{-}t\text{-}p}$.Garble: p_1 calls Π^1_{sh2PC}.Garble($\mathcal{C}_{kdc+record}$)*
2. *$\Pi_{g\text{-}t\text{-}p}$.Garble: p_2 calls Π^1_{sh2PC}.Evaluate($\mathcal{C}_{kdc+record}$)*
3. *$\Pi_{g\text{-}t\text{-}p}$.Prove: $\Pi_{comp}(\Pi^2_{sh2PC}$, $(\mathcal{C}_{kdc+record} + \mathcal{C}_{open} + \mathcal{C}_{\phi})$, $\Pi_{com})$*

under the assumptions that

- *in Π^1_{sh2PC} p_2 acts as the evaluator and p_1 acts as the garbler*
- *in Π^2_{sh2PC} p_1 acts as the evaluator and p_2 acts as the garbler*

- Π_{auth} *initially authenticates* Π_{com}

we say that malicious security holds for the garble-then-prove paradigm with a semi-honest 2PC *system in the garble phase.*

Proof 3.1: The adversary $\mathcal{A}$ is able to maliciously garble Π^1_{sh2PC} and obtain secrets from p_2. However, due to the asymmetric privacy setting established during the proving phase, $\mathcal{A}$ learns nothing beyond what $\mathcal{A}$ would have learned during the proving phase. And, a malicious garbling of $\mathcal{A}$ is recorded at p_2 because p_2 obtains all outputs of 2PC circuits executed in the garble phase. Thus, once the construction proceeds to step (3), and $\mathcal{A}$ has cheated, p_2 is able to detect it in step (9) of the Π_{comp} construction and can abort the protocol. This conditional abort option prevents $\mathcal{A}$ from obtaining a false provenance attestation of TLS data.

B. Extended Benchmarks

This chapter outlines the missing details of how we collect and compute evaluation tables. The extended benchmarks concern PET computations for the contributions *Janus* and *Origo*.

B.1. *Janus* Extra Benchmarks

The following subsection provides additional benchmarks and displays the entire E2E constructions of TLS oracles with *Janus* optimizations.

B.1.1. Microbenchmarks of 2PC Circuits

We present micro-benchmarks of secure computation building blocks in the Table B.1. This table compares circuit complexities, execution times, and communication overhead of 2PC circuits, where execution times and communication overhead are further divided into offline and online benchmarks. The 2PC circuits $\mathcal{C}_{\text{XHTS}}$ and $\mathcal{C}_{\text{k,iv}}$ derive session secrets in milliseconds and compute CBs via the circuit $\mathcal{C}^{\text{X}}_{\text{CB}_{2+}}$ for a 2 kB record in 164.9 milliseconds. An interesting fact to notice is that the AEAD tag circuit $\mathcal{C}_{tag}$ is efficient for small request sizes and scales sufficiently but not ideally for larger request sizes. The overhead in the circuit $\mathcal{C}_{tag}$ is introduced by the algebraic structure of the Galois field polynomials in GF(2^{128}), which, as an algebraic structure, is in conflict with the binary representation of computation in boolean GCs. The related works [11], [119] propose a scalable OT-based computation of the AEAD tag, which we consider as future work to improve our implementation.

Concerning data opening times, we can see that the *transparent* mode with the circuit $\mathcal{C}_{\text{tpOpen}}$ is more efficient compared to the *privacy-preserving* mode with the circuit $\mathcal{C}_{\text{zkOpen}}$. This behavior is expected because, the 2PC circuit of the transparent mode does not include the ciphertext, SHTS, and CB_{tag} verification inside the circuit (cf. Figure 4.8). As a consequence, the data communicated in the OT scheme of the *transparent* mode is about half the size of the *privacy-preserving* mode. The effect is further visible in the online communication cost, where the *transparent* mode communicates 3x less data than the *privacy-preserving* opening mode. As another reference benchmark (cf. f_ϕ of the last row in Table B.1), we evaluate the verification of a confidential document hash $H(f)$ in the circuit $\mathcal{C}_{\text{zkOpen}}$. To do so, we set the function $f_\phi = H(f) \stackrel{?}{=} H(\mathbf{pt})$ to a hash check on the 2 kB response data, with H=SHA256. Concerning online execution times, the extra hash evaluation yields a negligible overhead for the *client* but increases the communication overhead by a factor of 1.3x.

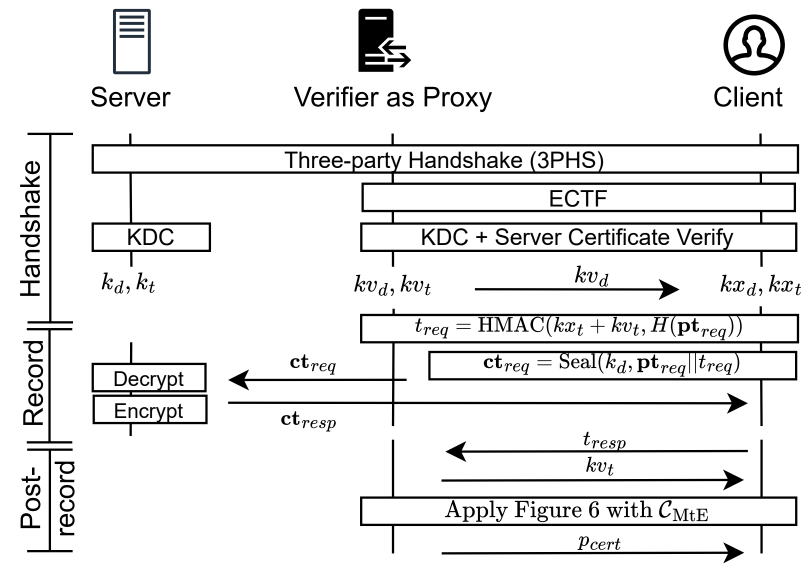

Server
Verifier as Proxy
Client
Handshake
Record
Post-record
Three-party Handshake (3PHS)
ECTF
KDC
KDC + Server Certificate Verify
k_d, k_t
kv_d, kv_t
kv_d
kx_d, kx_t
$t_{req} = \mathrm{HMAC}(kx_t + kv_t, H(\mathbf{pt}_{req}))$
$\mathbf{ct}_{req}$
$\mathbf{ct}_{req} = \mathrm{Seal}(k_d, \mathbf{pt}_{req}||t_{req})$
Decrypt
Encrypt
$\mathbf{ct}_{resp}$
t_{resp}
kv_t
Apply Figure 6 with $\mathcal{C}_{\mathrm{MtE}}$
p_{cert}

Figure B.1.: End-to-end protocol of a TLS 1.2 oracle running the first *Janus* optimization. Boxes crossing two vertical lines indicate 2PC protocols.

B.1.2. Complete E2E Janus Protocol

To simplify the reproduction of this work, we provide complete E2E descriptions of TLS oracles integrating the *Janus* optimizations. We consider a simple scenario where a single request and response are exchanged during the TLS session.

Concerning TLS 1.2, we describe a TLS oracle configured with TLS MtE (e.g. CBC-HMAC). This configuration performs best due to the constant-size 2PC circuits of the record phase (cf. Section 4.2.2). We depict the full E2E protocol of a TLS 1.2 oracle using the first *Janus* contribution in Figure B.1. TLS 1.3 oracles are configured with AEAD cipher suites. We depict a full E2E protocol of a TLS 1.3 oracle using both *Janus* optimizations in Figure B.2. Notice that for the computation of $\mathcal{C}_{\mathbf{zkOpen}}$, the key shares kv, kx indicate the secrets s_1, s_2 of the circuit (cf. Figure 4.8).

B.2. *Origo* Extra Benchmarks

Additional benchmarks of our work *Origo* show how ZKP sub-circuits influence the overall ZKP computation complexity.

B.2.1. Evaluation Details of End-to-end Computations.

We start by providing ZKP circuit benchmarks in Table B.2, where all provided values have been generated with our publicly shared repository[1]. The table compares ZKP oracle circuits as well as sub-circuits and differentiates benchmarks into online and offline computation

[1] `https://github.com/jplaui/gnark_lib`

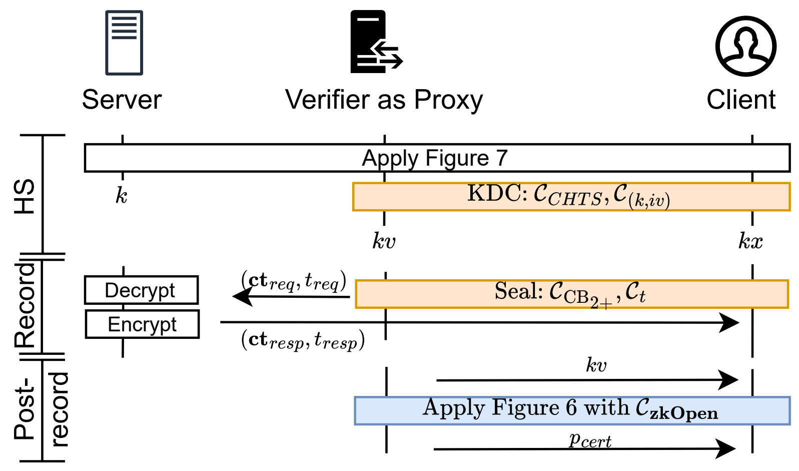

Server
Verifier as Proxy
Client
HS
Record
Post-record
Apply Figure 7
k
KDC: $\mathcal{C}_{CHTS}, \mathcal{C}_{(k,iv)}$
kv
kx
Decrypt
Encrypt
$(\mathbf{ct}_{req}, t_{req})$
Seal: $\mathcal{C}_{\mathrm{CB}_{2+}}, \mathcal{C}_t$
$(\mathbf{ct}_{resp}, t_{resp})$
kv
Apply Figure 6 with $\mathcal{C}_{\mathbf{zkOpen}}$
p_{cert}

Figure B.2.: End-to-end protocol of a TLS 1.3 oracle running both *Janus* optimizations. Boxes crossing two vertical lines indicate 2PC protocols. Orange boxes belong to the garble phase and the blue box indicates the prove phase.

and communication parameters. The separation is necessary to merge ZKP benchmarks with 2PC benchmarks for system end-to-end evaluation results. With respect to computing end-to-end benchmarks, we applied the following additional considerations. Execution times are measured in seconds and we group the ZKP circuit compile function and the ZKP setup function under offline execution times. Whereas prove and verify function calls count as online executions as these functions are executed at the prover and verifier when computing and verifying an oracle proof. Communication parameters are gained by deserializing *gnark* data structures except for the public witness parameters. The public witness of the key derivation circuit KDC are 32-byte variables $\mathbf{h}^{\mathrm{HS},opad}$, $\mathbf{MS}^{in}$, $\mathbf{SATS}^{in}$, and $\mathbf{SATK}^{in}$. The public witness of the tag variable consists of two encrypted counter blocks of size 16 bytes and the initialization vector of 12 bytes. The public input of the record circuit has a 32-byte ciphertext and depends on five 64-bit integers which sum to a total of 40 bytes.

Table B.1.: Secure computation benchmarks to implement the *Janus* optimizations. We allocate values to offline/online execution and communication columns. Dashed lines indicate which circuits belong to the handshake, record, and post-record phases. The rows 1,2 use are maliciously-secure 2PC benchmarks averaged over 10 executions using the framework *emp-ag2pc*. Other benchmarks have been collected with the framework Go-based 2PC framework *mpc*.

2PC Circuit	**Constraints** (x10^6)	**Execution Offline**	**Execution Online**	**Communication Offline**	**Communication Online**
SHA256	-	59.4 ms	10.09 ms	12.16 MB	13.184 kB
AES128	-	55.5 ms	2.8 ms	4.33 MB	6.8 kB
ECTF	-	-	212.96 ms	-	1.861 kB
$\mathcal{C}_{\text{XHTS}}$	3.14	215.56 ms	144 ms	34 MB	110 kB
$\mathcal{C}_{\text{k}^{m_1},\text{iv}}$	10.34	723.96 ms	484.82 ms	108.08 MB	356 kB
$\mathcal{C}^{\text{256 B}}_{\text{ECB}_{2+}}$ / $\mathcal{C}^{\text{2 kB}}_{\text{ECB}_{2+}}$	1.16 / 9.18	67.78 / 578.76 ms	67.6 / 164.9 ms	10.12 / 86.02 MB	116 / 566 kB
$\mathcal{C}^{\text{256 B}}_{tag}$ / $\mathcal{C}^{\text{2 kB}}_{tag}$	4.04 / 29.01	285.98 ms / 2.42 s	492.24 ms / 3.78 s	52.06 / 378.02 MB	512 kB / 2 MB
$\mathcal{C}^{\text{256 B q, 2 kB r}}_{\text{tpOpen}}$	12.69	0.89 s	0.46 s	126.01 MB	583 kB
$\mathcal{C}^{\text{256 B q, 2 kB r}}_{\text{zkOpen}}$ / f_ϕ	12.73 / 17.15	0.89 / 1.13 s	2.04 / 2.08 s	127.02 / 168.03 MB	2.13 / 2 MB

Table B.2.: ZKP benchmarks averaged over 10 executions using the framework *gnark*. We classify benchmarks into offline and online execution and communication values such that merging ZKP numbers with 2PC numbers for system end-to-end values becomes feasible. We use the abbreviations Compiled Constraint System (CCS), Prover Key (PK), and Verifier Key (VK), Public Witness (PW), and proof π, where PK and VK together form the *common reference string* of the ZKP system.

		Execution Offline		Execution Online		Communication Offline			Communication Online	
ZKP Circuit	# Constraints (x10^3)	Compile (s)	Setup (s)	Prove (s)	Verify (s)	CCS, SRS (MB)	PK (MB)	VK (KB)	π (KB)	PW (B)
					Groth16 backend					
KDC	322.4	0.9	11.12	0.48	0.001	35.3	47.8	4.93	0.128	128
Tag	285.8	1.28	9.17	0.6	0.001	106.6	40.1	2.37	0.128	44
Record	288.1	1.31	9.24	0.61	0.001	105.1	40.4	2.02	0.128	72
Commit	44.5	0.12	1.5	0.07	0.001	4.31	6.35	1.34	0.128	32
CommitRecord	332.7	1.37	10.37	0.61	0.001	113.56	44.73	3.04	0.128	104
CommitTagRecord	618.6	2.67	20.16	1.21	0.001	223.61	84.9	5.09	0.128	148
KDCTagRecord	896.4	3.43	25.35	1.21	0.001	255.93	111.68	8.16	0.128	244
					Plonk backend					
KDC	573.0	7.64	6.08	7.38	0.002	52.7, 33.5	327.1	0.54	0.552	128
Tag	551.9	7.65	6.14	7.46	0.002	61.5, 33.5	327.1	0.54	0.552	44
Record	556.5	7.83	6.16	7.67	0.002	61.9, 33.5	327.1	0.54	0.552	72
Commit	76.7	1.01	0.86	1.04	0.002	6.4, 4.1	40.8	0.54	0.552	32
CommitRecord	634.3	7.78	6.03	7.41	0.002	69.3, 33.5	327.1	0.54	0.552	104
CommitTagRecord	1186.2	15.69	12.36	14.95	0.002	131.7, 67.1	654.3	0.54	0.552	148
KDCTagRecord	1681.4	15.91	11.9	14.75	0.002	177.9, 67.1	654.3	0.54	0.552	244

C. Evolution of Zero-knowledge Proofs

During the composition of this thesis, the field of ZKPs evolved from dedicated proof systems to systems capable of proving arbitrary statements. Since our work makes use of both dedicated and general-purpose proof systems, this chapter outlines the evolution concisely.

C.1. Dedicated Zero-knowledge Proofs

In our work *A-PoA*, we rely on a dedicated ZKP system with the ability to prove a fixed computational relation, namely the discrete log of the RSA accumulator. In the following, we highlight how dedicated proof systems function. Subsequently, we explain the dedicated proof system used in our construction called *A-PoA*.

Proving Knowledge of the Schnorr Discrete Log

The PoK protocol for the discrete log problem of Schnorr et al [153], shown in Figure C.1, is an example of an interactive protocol (also referred to as Σ-protocol) which leverages a commitment scheme to let a prover P convince a verifier V. In a commitment scheme, P sends a commitment value $R = g^r$ (commitment) to V without revealing the secret r. After receiving a challenge from V (commitment challenge), at a later point in time, P sends a seed value (commitment response), such as $z = c \cdot \alpha + r$ to prove the factoring assumption of the commitment which V verifies in Equation C.1 [154]. Values of U (basis of discrete log relation) and g (basis to encrypt commitment) are drawn from a group $\mathbb{G}$ of prime numbers p and shared between P and V.

A PoK allows a *prover P* to convince a *verifier V* that P knows the solution of a hard problem without revealing the solution. We say that the conversation between P and V is a PoK of relation $\mathcal{R}$ when the following properties hold: (1) Completeness, (2) Soundness, and (3) Zero-knowledge [155].

Prover		**Verifier**
$\alpha \in \mathbb{Z}_q$		$(U, A = U^\alpha) \in \mathbb{G}$
$r \leftarrow \mathbb{Z}_q$	$\xrightarrow{R = g^r \in G}$	
	$\xleftarrow{c \in C}$	$c \leftarrow C = \{1, \ldots, 2^\lambda\} \subseteq \mathbb{Z}_q$
	$\xrightarrow{z = c \cdot \alpha + r \in \mathbb{Z}}$	accept iff $U^z = A^c \cdot R$

Figure C.1.: The Schnorr Proof of Knowledge protocol, as shown in [45].

Completeness: Completeness means that P, knowing the solution, can successfully convince V. Originally, completeness was solved iteratively through an interactive protocol between P and V. Here, P answers a challenge of knowledge correctly multiple times to convince V. From V's perspective, if the likelihood L of answering a challenge correctly is $L \leq \frac{1}{2}$, the confidence of V increases if P knows/provides a convincing argument multiple rounds.

$$U^z = g^z = g^{c \cdot \alpha + r} = g^r \cdot (g^\alpha)^c = R \cdot (U^\alpha)^c = R \cdot A^c \tag{C.1}$$

Soundness: Soundness means that P, not knowing a solution, will fail to convince V. The typical way to prove the soundness of a protocol is to leverage the concept of the *knowledge extractor*. An *extractor* with the ability to extract secret knowledge from P convinces V of the existence of a satisfying solution. In order to extract the unknown secret value from P, the *extractor* is given rewind access to the actions of P. Such an ability can achieve two accepting conversations for one given verification key, effectively enabling an adversary to break the discrete-log assumption with non-negligible probability.

The important feature of the *extractor* concept is the assumption that V has access to a property that it does not have under real circumstances. In order to extract the unknown secret value from the P, the extractor is given rewind access to the P. Through this ability we achieve two accepting conversations for one given verification key, effectively enabling an adversary to break the discrete-log assumption.

Take, for example, the Schnorr protocol of Figure C.1 and the assumption that V is able to receive two accepting conversations (R, c, z) and (R, c', z'). The case that P sends the same R twice to V does not happen in a real application of this protocol. The *extractor* model requires such an assumption which allows V to extract the witness α from P with Equation C.2.

$$\begin{aligned} \text{with } \; U^z = A^c \cdot R \quad &\text{and} \quad U^{z'} = A^{c'} \cdot R \\ \Longrightarrow \qquad & U^{z-z'} = A^{c-c'} \\ \Longrightarrow \qquad & \alpha = \frac{z - z'}{c - c'} = \log_g(U) \end{aligned} \tag{C.2}$$

Figure C.1 summarizes this interaction which has the property of HVZK. HVZK means that witness extraction is possible if the verifier leverages two accepting conversations. Here, the verifier requests the same R twice and returns c and c' in each round respectively with $c \neq c'$. Since $c - c'$ is invertible in $\mathbb{Z}_q$, the verifier is able to calculate $U^{z-z'} = V^{c-c'}$, $U^{(z-z')/(c-c')} = V$, and, as a result, $\alpha = (z - z')/(c - c')$.

HVZK: HVZK in a PoK scheme requires V to learn nothing but the validity of a convincing assertion of P. From the perspective of V, this means that for the case of a true/valid assertion of P, V does not learn anything from the interaction with P that V could have learned on her own. To prove ZK, a concept of a *Simulator* is used to simulate the protocol p_{sim} between P and V. Taking the definition of a valid transcript as the output of the interactive protocol between V and P with a valid assertion, the *Simulator* produces a simulated transcript T_{sim}

which can be compared to the transcript T_{real} of the real interactive protocol p_{real} [156]. Note that the transcripts are viewed from the perspective of V.

If the distributions of T_{sim} and T_{real} remain statistically indistinguishable, a computationally unbounded adversary in the form of V is not able to extract/learn any information by participating in each of the protocols p_{sim} and p_{real} [157]. The reason for this is that T_{sim} can be modeled such that it does not contain any secret knowledge of P, enabling a statistically indistinguishable simulation of the real protocol. Since T_{sim} is statistically almost identical to T_{real}, T_{real} does not reveal any additional information. The calculation of T_{sim} of the protocol of the discrete log problem is shown in Equation C.3.

$$\begin{aligned} \text{Select } & z, c \leftarrow \mathbb{Z}_q \\ \text{Calculate } & \alpha = \frac{g^z}{U^c} \\ \text{Output } & T_{sim} = (\alpha, c, z) \end{aligned} \tag{C.3}$$

To achieve almost identical transcript distributions, note that c is almost uniform random and α is almost independent of the conversation history [158] [159].

From the information-theoretic perspective and with the definition of the distributions of the transcript, it is possible to introduce the three different definitions of ZK. Perfect ZK implies that the distributions of T_{sim} are exactly equal to T_{real}, whereas Statistical Zero-Knowledge (SZK) requires the distributions to be statistically almost identical. SZK relies on the language of promise problems where the problem of almost identical distributions can be reduced to and expressed by Statistical Difference (SD) [157]. Computational ZK requires computational indistinguishability of the distributions T_{sim} and T_{real}. This means that no efficient algorithm is able to notice a difference between these almost identical distributions with a non-negligible probability.

A caveat to the protocol of PoK of the discrete log of Figure C.1 is that it is HVZK. HVZK requires V to behave according to the protocol and allows a *Simulator* to model the actions of V with deterministic behavior.

Zero-knowledge: Zero-knowledge provides robustness against a cheating verifier. The work of Sahai et al. [160] proves that every HVZK protocol can be transformed into a general SZK protocol with a cheating verifier. To do so, it is necessary to computationally bind the challenge of the verifier to a commitment value before the verifier receives the commitment of P [155]. This measure prevents the verifier from guessing the distribution of the commitment of P which, in the case of a correct guess, extracts knowledge of the subsequent transcript of the protocol. Extracting knowledge throughout the protocol contradicts the original idea of ZK. Malicious verifier resistance affects the *Simulator* concept which must adapt to the situation by guessing the challenge of the verifier before the commitment computation of the verifier [161].

Non-interactivity: Non-interactivity of the Schnorr protocol in Figure C.1 together with the general ZK property can be achieved by leveraging the Fiat-Shamir heuristic [103]. To do so, it is necessary to replace the random challenge c of V with a hash function computation of the commitment value $c = H(R)$, where $R = g^r$. This calculation can be done by P which reduces

```
zk.Prove($w_x, x, a_t$):
1. $k, \rho_x, \rho_k \overset{R}{\leftarrow} [-B, B]$;                    $z = g^x h^{\rho_x}$
2. $A_g = g^k h^{\rho_k}$;                                   $A_{w_x} = w_x^k$
3. $l \leftarrow H_{\text{prime}}(w_x, a_t, z, A_g, A_{w_x})$;     $c \leftarrow H(l)$
4. $q_x \leftarrow \lfloor (k + c \cdot x)/l \rfloor$;                 $q_\rho \leftarrow \lfloor (\rho_k + c \cdot \rho_x)/l \rfloor$
5. $r_x \leftarrow (k + c \cdot x) \bmod l$;                  $r_\rho \leftarrow (\rho_k + c \cdot \rho_x) \bmod l$
6. return: $\pi \leftarrow \{l, z, g^{q_x} h^{q_\rho}, w_x^{q_x}, r_x, r_\rho\}$
zk.Verify($w_x, a_t, \pi$):
1. $\{l, z, Q_g, Q_{w_x}, r_x, r_\rho\} \leftarrow \pi$;              $c = H(l)$
2. $A_g \leftarrow Q_g^l g^{r_x} h^{r_\rho} z^{-c}$;                 $A_w \leftarrow Q_{w_x}^l w_x^{r_x} a_t^{-c}$
3. Verify $r_x, r_\rho \in [l]$;                         $l = H_{\text{prime}}(w_x, a_t, z, A_g, A_w)$
4. return: $\{0, 1\}$
```

Figure C.2.: The NI-ZKPoKE zk.Prove and zk.Verify algorithms as defined in the work [45].

the communication overhead of the interactive protocol to a single round. The requirements (uniform random distribution, statistical independence) of the challenge c apply to the hash function in use.

The *Simulator* computation changes respectively. The computation stays the same as in Equation C.3 except that an additional requirement of $c = H(T)$ applies. The Fiat-Shamir computation provides soundness under the random oracle model and its application in the standard model only provides security for special protocols based on elliptic curves, factoring, and quadratic residues [162].

Dedicated ZKP for A-PoA

The problem with the above-described proof of discrete log is that it cannot be applied to the cryptographic accumulator definition which we use in our work *A-PoA*. The reason is that the Schnorr proof of discrete log works for generators $U \in \mathbb{G}_q$ of a finite cyclic group of prime order q. However, the RSA accumulator used in *A-PoA* requires a generator $g_{\text{acc}} \overset{R}{\leftarrow} \mathbb{Z}_n^* \backslash \{\pm 1\}$, with $n = p \cdot q$, $p = 2 \cdot p' + 1$, and $q = 2 \cdot q' + 1$, where p' and q' are *Sophie Germain* primes (p such that $2p + 1$ is prime). In order to find *Sophie Germain* primes, safe primes with bit size k =512, 768, 1024, 2048, 4096 must be computed repeatedly, such that finding a random prime p (k-1 bits) with $2p + 1$ as another prime is possible (or repeatedly find a random k-bit prime p, until $(p$-1$)/2$ is prime). The *Miller-Rabin* primality test can be used to find such primes faster.

As a solution, the work [45] constructs a NI-ZKPoKE of discrete log in a group of unknown order $\mathbb{G}_?$ with security under the adaptive root problem (cf. Figure C.2). This ZKP can be applied to the RSA accumulator $a_t = (g_{\text{acc}})^2 \mod n$, with $g_{\text{acc}} \overset{R}{\leftarrow} \mathbb{G}_?$. Remember that accumulator elements compute as $x \in$ *primes*, with $x \neq p', q'$ and $A \leq x \leq B$, where A, B can be chosen with arbitrary polynomial dependence (linear independence of polynomials,

Extraction:

1. Extract x, ρ such that $z = g^x \cdot h^\rho$ and $g^{s_1} \cdot h^{s_2} = A_g \cdot z^c$
2. Set $R \leftarrow \{\}$ and sample $s_1, s_2 \xleftarrow{R} [0, 2^\lambda]$
3. Sample $l \xleftarrow{R} \text{Primes}(\lambda)$, $c \xleftarrow{R} [0, 2^\lambda]$ and send s_1, s_2, l, c to $\mathcal{A}_1$
4. Get output Q_g, Q_u, r_1, r_2 from $\mathcal{A}_0$. If transcript is accepting ($Q_g^l \cdot g^{r_1} \cdot h^{r_2} = A_g \cdot z^c$ and $Q_u^l \cdot u^{r_1} = A_u \cdot w^c$), then update $R \leftarrow R \cup \{(r_1, r_2, l, c)\}$. Otherwise go to step 2.
5. Use CRT to compute $s_1 = r_1^{(i)} \mod l^{(i)}$ and $s_2 = r_2^{(i)} \mod l^{(i)}$, for each $(r_1^{(i)}, r_2^{(i)}, l^{(i)}) \in R$. If $u^{s_1} = A_u \cdot w^c$ then output s_1, otherwise return to step 2.
6. Repeat for s_1', s_2', c', so that $x = \Delta s_1 / \Delta c = (s_1 - s_1')/(c - c')$ and $\rho = \Delta s_2 / \Delta c = (s_2 - s_2')/(c - c')$, with extraction based on $u^{s_1} = A_u \cdot w^c$ and $u^{s_1'} = A_u \cdot w^{c'}$, thus $(u^x)^{\Delta c} = w^{\Delta c} \Rightarrow u^x = w$.

Simulation:

1. $\widetilde{c} \xleftarrow{R} [0, 2^\lambda]$, $\widetilde{l} \xleftarrow{R} \text{Primes}(\lambda)$
2. $\widetilde{z} \leftarrow h^{\widetilde{\rho}}$, $\rho \xleftarrow{R} [B]$
3. $\widetilde{q}_x, \widetilde{q}_r \xleftarrow{R} [B]^2$
4. $\widetilde{r}_x, \widetilde{r}_\rho \in [l]^2$
5. $\widetilde{Q}_g \leftarrow g^{\widetilde{q}_x} \cdot h^{\widetilde{q}_\rho}$, $\widetilde{Q}_u \leftarrow u^{\widetilde{q}_x}$
6. $\widetilde{A}_g \leftarrow \widetilde{Q}_g^l \cdot g^{\widetilde{r}_x} \cdot h^{\widetilde{r}_\rho} \cdot z^{-\widetilde{c}}$, $\widetilde{A}_u \leftarrow \widetilde{Q}_u^l \cdot u^{\widetilde{r}_x} \cdot w^{-\widetilde{c}}$

Figure C.3.: Extraction and simulation algorithms to show the soundness of the zero-knowledge property of the NI-ZKPoKE construction of the work [45]. The witness extraction depends on the Chinese Remainder Theorem (CRT).

arbitrary constant coefficients, distinct positive integers as grades), respecting the security parameter λ, as long as $2 < A$ and $B < A^2$. The generation of $x \leftarrow H_{\text{prime}}(x', \lambda)$ using H_{prime} delivers collision resistant accumulator elements.

Completeness of the NI-ZKPoKE construction computes according to the equations C.4.

$$\begin{aligned} Q_g^l \cdot g^{r_x} \cdot h^{r_\rho} &= (g^{q_x} \cdot h^{q_\rho})^l \cdot g^{r_x} \cdot h^{r_\rho} = g^{q_x \cdot l + r_x} \cdot h^{q_\rho \cdot l + r_\rho} = g^{s_x} \cdot h^{s_\rho} \\ &= g^{k + c \cdot x} \cdot h^{\rho_k + c \cdot \rho_x} = g^k \cdot h^{\rho_k} \cdot g^{x \cdot c} \cdot h^{\rho_x \cdot c} = A_g \cdot z^c \\ \Rightarrow Q_u^l \cdot u^{r_x} &= (u^{q_x})^l \cdot u^{r_x} = u^{q_x \cdot l + r_x} = u^{s_x} = u^{k + c \cdot x} = u^k \cdot u^{x \cdot c} = A_u \cdot w^c \end{aligned} \tag{C.4}$$

The *soundness* and *zero-knowledge* properties hold according to the extraction and simulation algorithms depicted in Figure C.3. In the simulation paradigm, statistical indistinguishability holds between $(\widetilde{z}, \widetilde{A}_g, \widetilde{A}_u, \widetilde{c}, \widetilde{l}, \widetilde{Q}_g, \widetilde{Q}_w, \widetilde{r}_x, \widetilde{r}_\rho)$ and $(z, A_g, A_u, c, l, Q_g, Q_w, r_x, r_\rho)$.

C.2. General-purpose Zero-knowledge Proof Systems

After the advent of dedicated proof systems, the advancements of ZKP technology led to the invention of efficient, general-purpose proof systems [17]. General-purpose proof systems

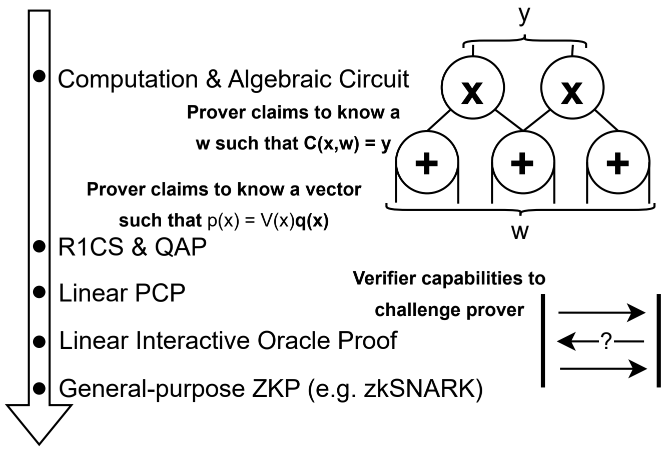

y
Computation & Algebraic Circuit
Prover claims to know a w such that C(x,w) = y
x
x
+
+
+
w
Prover claims to know a vector such that p(x) = V(x)q(x)
R1CS & QAP
Linear PCP
Verifier capabilities to challenge prover
?
Linear Interactive Oracle Proof
General-purpose ZKP (e.g. zkSNARK)

Figure C.4.: Sequence of building blocks for the construction of general-purpose ZKPs.

are capable of proving arbitrary computation logic and power the works *Portal*, *Origo*, and *zkGen*. The following paragraphs introduce the main building blocks of general-purpose proof systems.

Building Blocks

The construction of a proof system depends on the interplay of multiple compatible building blocks.

Arithmetic Representations & Circuit Satisfiability: In practice, the computation logic of general-purpose proof systems is created using a DSL [98], which can be translated to an algebraic circuit (cf. Figure C.4). For the algebraic circuit, it must hold that a witness satisfies the circuit logic. The witness has two parts, where the private witness is known by the prover and the public part is known by the verifier. Due to the fact that the algebraic circuit cannot be proven using cryptography, general-purpose proof systems translate the algebraic representation of the computation over a constraint system (e.g. R1CS, AIR, Plonkish) into a Quadratic Arithmetic Program (QAP).

Quadratic Arithmetic Program: QAPs are systems of equations over polynomials, which encode the problem defined by the algebraic circuit into a representation that can be proven using cryptography. The functions and parameters expressing the QAP can be encrypted using functional commitment schemes and can be queried by the verifier according to an Interactive Oracle Proof (IOP) model.

Functional Commitment Schemes: Functional commitment schemes are cryptographic objects that allow the commitment to a vector of messages instead of committing to a single message instance (cf. Section 2.2.2). Using functional commitments, committed polynomials can be shared by the prover towards the verifier. For instance, to commit to a polynomial, the commit function encrypts all coefficients of the polynomial. Popular functional commitment schemes are of the type polynomial, multilinear, vector, or inner product. Popular mathematical constructions of functional commitments rely on hash functions, bilinear groups (e.g.

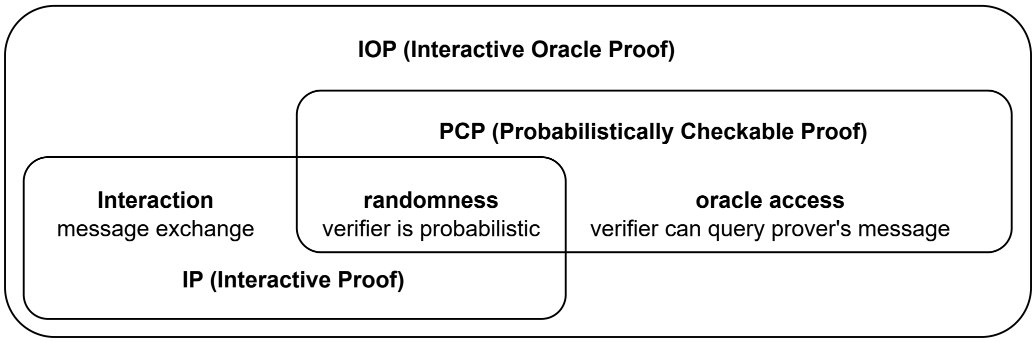

IOP (Interactive Oracle Proof)
PCP (Probabilistically Checkable Proof)
Interaction
message exchange
randomness
verifier is probabilistic
oracle access
verifier can query prover's message
IP (Interactive Proof)

Figure C.5.: Classification of information theoretic proofs based on the verifier capabilities.

KZG commitments), groups of unknown order, or elliptic curves. After the prover shares commitments, the verifier has the opportunity to evaluate committed functions at selected points (e.g. $f(u) \overset{?}{=} \text{v}$). The powers of the verifier are defined via different models. For instance, an interactive proof (IP) as depicted in Figure C.1 allows the verifier to randomly select c and exchange messages with the prover. In contrast, a linear Probabilistic Checkable Proof (PCP) is a vector of elements in a finite field on which the verifier can perform linear operations. In other words, the response from the oracle (the prover) to a verifier query is a linear function.

Compatible Interactive Oracle Proofs: Interactive oracle proofs (IOPs) are information-theoretic constructs which assume a verifier with oracle access, randomness, and interaction capabilities (cf. Figure C.5). Interaction means that the verifier can exchange messages with the prover. Randomness allows the verifier to behave probabilistic. Oracle access allows the verifier to query the messages sent by the prover. The functional commitment scheme must support the capabilities defined by the verifier model. Queries by the verifier can target individual points, polynomial evaluations (e.g. a polynomial zero test), tensor queries, linear queries, or more. The minimization of the checks at the verifier boosts the performance (efficient verifier complexity) of proof systems.

Types of Proof Systems & Key Properties

The freedom to combine the above-defined building blocks leads to many different forms of proof systems that exist today. In the following, we highlight the most popular types of proof systems and specify which ones we use in our contributions.

- **zkSNARKs** are zero-knowledge succinct non-interactive arguments of knowledge. zkSNARKs are characterized by succinctness, which yields short proofs and efficient verification times that behave sublinear with regard to the proving complexity. We use zkSNARKs in our contributions *Origo*, *Portal*, and *zkGen*.
- **zkSTARKs** are scalable, transparent arguments of knowledge. The key characteristic of STARKs is transparency which removes the requirement of a trusted setup assumption.

- **MPC-based ZKPs** achieve efficient evaluations of non-algebraic statements. We use a transparent MPC-based proof system in our work *Janus*.
- **Recursive ZKPs** enable the proving of proofs.
- **zkEVMs** are zero-knowledge virtual machines that are characterized by efficient zero-knowledge proving of any type of computation.

D. Publications in This Thesis

Major parts of this thesis have undergone a peer-reviewed publication procedure at international research conferences. In the following, we highlight where and how our publications contributed to the contents of this thesis. We present our publications in chronological order.

A-PoA: Anonymous Proof of Authorization for Decentralized Identity Management. Published in: *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, Sydney, Australia, 03-06 May 2021. Pages: 9. Authors: **J. Lauinger**, *J. Ernstberger, E. Regnath, M. Hamad, S. Steinhorst*

In this work [83], we analyze privacy-preserving ways to delegate the authority of access to another party in the ecosystem of decentralized identifiers. The contribution in this work touches on the topics of ZKPs. We use a dedicated ZKP system to prove the access authority via a privacy-preserving membership proof. The contents and findings of this work contribute to the contents of Section 3.3 and Appendices A and C of this thesis.

SoK: Data Sovereignty. Published in: *IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, Delft, Netherlands, 03-07 July 2023. Pages: 21. Authors: *J. Ernstberger,* **J. Lauinger**, *F. Elsheimy, L. Zhou, S. Steinhorst, R. Canetti, A. Miller, A. Gervais, D. Song*

In this work [84], we analyze the notions of data sovereignty which can be constructed today. To construct data sovereignty from the device perspective, we assume that devices have access to cryptographic material in the form of key pairs. Further, we investigate advanced cryptographic techniques and PETs (e.g. MPC, ZKPs, etc.) to protect data and accounts during data administration, sharing, and outsourced processing. Minor parts of this work contribute to the motivation in Chapter 1, and to Sections 3.1 and 5.2 of this thesis.

zkGen: Policy-to-Circuit Transpiler. Published in: *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, Dublin, Ireland, 27-31 May 2024. Pages: 5. Authors: **J. Lauinger**, *J. Ernstberger, and S. Steinhorst*

In this work [109], we automate the policy-driven automation of ZKP circuits. The proposed functionalities let devices dynamically react to any privacy-preserving computation requirements such that the provision of PET computation remains in the hands of devices instead of specialized services. The contents of this work have been used to create the contents in Section 4.4 of this thesis.

Portal: Time-Bound and Replay-Resistant Zero-Knowledge Proofs for Single Sign-On. Published in: *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, Dublin, Ireland, 27–31 May 2024. Pages: 7. Authors: **J. Lauinger**, *S. Bezmez, J. Ernstberger, and S. Steinhorst*

In this work [82], we solve a security issue that concerns the ZKP verification at smart contracts. Based on our findings, we propose a SSO alternative based on a sovereign, cost-efficient identity system. This contribution enables merging our contributions A-PoA, *Janus* or *Origo*, and *Portal* into a single system with strong data sovereignty guarantees. The findings of this work have been used to create the contents in Chapters 1 and 5, Section 3.2, and the Appendices A and C of this thesis.

Janus: Fast Privacy-Preserving Data Provenance For TLS. Published in: *The 25th Privacy Enhancing Technologies Symposium (PETS25 Issue 1)*, Washington DC, USA, 14–19 July 2025. Pages 20. Authors: **J. Lauinger**, *J. Ernstberger, A. Finkenzeller, and S. Steinhorst*

In this work [58], we optimize the proof computations of clients in protocols that verify the provenance of data. The insights of this work contribute to an efficient reclaiming of data ownership for devices that store, maintain, and process data at external parties. The findings of this work contribute to the creation of Chapters 2 and 5, Sections 4.1 and 4.2, and Appendices A and B of this thesis.

ORIGO: Proving Provenance of Sensitive Data with Constant Communication. Published in: *The 25th Privacy Enhancing Technologies Symposium (PETS25 Issue 2)*, Washington DC, USA, 14–19 July 2025. Pages 18. Authors: *, J. Ernstberger**, **J. Lauinger***, *Y. Wu, A. Gervais, and S. Steinhorst*

In this work [97], we optimize protocols that verify the provenance of secure channel data with respect to network bandwidth requirements. Our solution entirely removes the overhead of secure 2PC computations and enables a deployment of TLS oracles at an Internet scale. The findings of this work contribute to the contents in Sections 4.1 and 4.3, and Appendix B.

List of Figures

List of Tables

Acronyms

1-RTT One Round-trip Time. 29, 73, 78, 82, 96, 115, 123

3PHS Three-party Handshake. 30, 76, 81, 83, 88, 97, 105, 122–124, 141

AE Authenticated Encryption. 29, 77, 143

AEAD Authenticated Encryption with Associated Data. 19, 20, 29, 74, 78, 79, 81, 82, 85, 86, 88–90, 92, 98, 127, 128, 141

AES Advanced Encryption Standard. 17, 20, 78, 79, 88, 90, 141

AI Artificial intelligence. 72, 73, 94

A-PoA Anonymous Proof of Authorization. 55, 56, 58, 59, 62–65, 145

ASCII American Standard Code for Information Interchange. 110

BGP Border Gateway Protocol. 96

C2PA Coalition for Content Provenance and Authenticity. 94

CATS Client Application Traffic Secret. 100

CBC Cipher Block Chaining. 29, 88

CBC-HMAC Cipher Block Chaining Hash-based Message Authentication Code. 77, 78, 87–90, 128

CBs Counter Blocks. 20, 86, 92, 127

CCPA California Consumer Privacy Act. 2

CCS Compiled Constraint System. 130, 145

CD Credential Definition. 12, 54–57, 59

CDH Computational Diffie-Hellman. 16

CF Client Finished. 29, 84

CH Client Hello. 28–30, 73, 82, 83, 96, 97

CHTS Client Handshake Traffic Secret. 84

CIA Credential Issuing Authority. 12, 54–56, 58, 59, 61, 63–65

CORS Cross-Origin Resource Sharing. 70, 105, 117

CRT Chinese Remainder Theorem. 135, 144

CS Credential Schema. 12, 54–59, 62–65

DDH Decision Diffie-Hellman. 16

DHE Diffie–Hellman Exchange. 29, 30, 76, 77, 83, 100

DHKE Diffie-Hellman Key Exchange. 28

DID Decentralized Identifier. 11, 12, 53, 54, 58, 59, 61, 62, 65, 115, 120

DLP Discrete Logarithm Problem. 16

DNS Domain Name System. 8, 75, 96, 97

DPF Data Policy Framework. 2

DSA Digital Services Act. 2

DSL Domain Specific Language. 107, 109–111, 136

E2E End-to-end. 73, 74, 87, 90, 115, 127, 128

EC Clliptic Curve. 16, 30, 31, 77, 141

ECDHE Clliptic Curve Diffie–Hellman Exchange. 30, 88

ECDLP Elliptic Curve Discrete Log Problem. 16

ECTF Elliptic Curve to Field. 30, 31, 77, 81, 83, 88, 90, 141

eID Electronic Identity. 65

ENS Ethereum Name System. 11

EOA Externally Owned Account. 12

EU European Union. 2

EVM Ethereum Virtual Machine. 107

GCM Galois Counter Mode. 20, 29, 74, 78, 86, 87, 96, 141

GCs Garbled Circuits. 23, 26, 77, 127

GDPR General Data Protection Regulation. 2

GF Galois field. 75, 85

HE Homomorphic Encryption. 21, 113

HKDF HMAC-based Key Derivation Function. 19, 27

HMAC Hash-based Message Authentication Code. 27, 78, 99

HS Handshake Secret. 29, 83, 100, 102, 103, 129

HVZK Honest Verifier Zero-knowledge. 23, 25, 26, 73–76, 80–82, 85, 86, 88–90, 92, 115, 116, 121–123, 132, 133, 143

IdM Identity Management. 9, 54

IdP Identity Provider. 10, 39–41, 53, 142

IOP Interactive Oracle Proof. 136, 137

IoT Internet of Things. 5

IP Internet Protocol. 7, 8, 70, 75, 93, 97, 117

IPFS Interplanetary File System. 12, 13, 45–47, 49, 51

IPNS InterPlanetary Name System. 50

JSON JavaScript Object Notation. 110, 112

KYC Know Your Customer. 41

L1 Layer 1. 53

L2 Layer 2. 53

LAN Local Area Network. 90, 104, 105

LOC Lines of Code. 112

LWE Learning with Errors. 16

MITM machine-in-the-middle. 44, 70, 71, 75, 76, 90, 92, 95, 96, 104, 105, 115

MPC Multi-party Computation. 3, 5, 17, 67, 70, 95, 99, 113, 139

MT Merkle Tree. 24, 44, 50, 142

MtA Multiplicative to Additive. 21, 30, 90, 92, 93

MtE Mac-then-Encrypt. 77–79, 89, 90, 128

NI-ZKPoKE Non-interactive Zero Knowledge Proof of Knowledge of Exponent. 55, 56, 59–66, 114, 121, 134, 135, 142–144

OAuth 2.0 Open Authorization 2.0. 10, 38, 40

OBE On-Board Equipment. 55

OT Oblivious Transfer. 22, 23, 31, 32, 127

P2P Peer-to-peer. 13

PCP Probabilistic Checkable Proof. 137

PCS Polynomial Commitment Scheme. 26

PET Privacy Enhancing Technology. 3–6, 14, 34, 38, 67, 70, 71, 95, 106, 109, 113, 115, 116, 118, 127, 139

PIOP Polynomial Interactive Oracle Proof. 26

PK Prover Key. 130, 145

PKC Public Key Cryptography. 8, 11, 12, 14, 17, 24, 25, 42

PKI Public Key Infrastructure. 8, 9, 11, 14, 29, 66, 75, 96, 97, 99

PoK Proof of Knowledge. 21, 131–133

PoS Proof of Stake. 41, 43–45, 47, 50, 52, 120

PPT Probabilistic Polynomial Time. 23, 44, 119, 124

PRF Pseudo-random Function. 17

PRG Pseudo-random Generator. 17

PRP Pseudo-random Permutation. 17

PW Public Witness. 130, 145

QAP Quadratic Arithmetic Program. 136

RA Root Authority. 12, 54–56, 58, 59, 64, 65

RAM Random Access Memory. 49, 89

RSA Rivest–Shamir–Adleman. 16, 21, 60, 62–65, 114, 120, 121, 131, 134, 142, 143, 145

RSAP RSA Problem. 16

RTT Round-trip Time. 91, 104, 145

SATS Server Application Traffic Secret. 29, 100, 102, 129

SD Statistical Difference. 133

SDK System Development Kit. 50

SF Server Finished. 29, 83, 84, 97, 100, 102

SH Server Hello. 28, 30

SHTS Server Handshake Traffic Secret. 29, 33, 73, 74, 82–86, 89, 90, 97–100, 102, 103, 115, 121, 123, 125, 127

SIWE Sign-In with Ethereum. 11, 40, 41, 47

SNI Server Name Identifier. 97

SSH Secure Shell. 68

SSIM Self-Sovereign Identity Management. 54

SSO Single Sign-on. 1, 2, 6, 10, 11, 40, 41, 52, 114, 140

SZK Statistical Zero-Knowledge. 133

TCP Transmission Control Protocol. 93, 97

THF Trapdoor hash function. 93

TLS Transport Layer Security. 9, 15, 25, 27–30, 44, 49, 51, 68–90, 92, 93, 95–106, 109, 115, 117, 122–126, 128, 141, 143–145

UC Universal Composability. 18

URL Uniform Resource Locator. 46, 47

US United States. 2

VC Verifiable Credential. 11, 12, 53–56, 58, 59, 115, 141

VK Verifier Key. 130, 145

vMAC Message Authentication Code. 29, 33, 88

W3C World Wide Web Consortium. 11, 38, 54, 55, 115, 118

WAN Wide Area Network. 91, 104, 105, 145

ZK Zero-Knowledge. 63, 65, 132, 133

ZKP Zero-knowledge Proof. 4–6, 17, 24–26, 34, 37, 38, 40, 41, 43–53, 56, 67, 68, 70, 75, 78, 79, 85, 88, 89, 92, 93, 95, 96, 98–101, 103–109, 111–115, 117, 128–131, 134, 135, 138–140, 142–145

zkVM zero-knowledge Virtual Machine. 4

Bibliography

[1] H. Saleem and M. Naveed, "Sok: Anatomy of data breaches", *Proceedings on Privacy Enhancing Technologies*, 2020.

[2] E. Union, *Regulation Document 32016R0679*, `https://eur-lex.europa.eu/eli/reg/2016/679/oj`, [Online; accessed 07-August-2024].

[3] S. of California Department of Justice, *California Consumer Privacy Act*, `https://oag.ca.gov/privacy/ccpa`, [Online; accessed 07-August-2024].

[4] E. Union, *General Data Protection Regulation*, `https://gdpr-info.eu/`, [Online; accessed 07-August-2024].

[5] I. T. Administration, *Data Privacy Framework*, `https://www.dataprivacyframework.gov/EU-US-Framework`, [Online; accessed 07-August-2024].

[6] P. A. Cloudflare, *All you need to know about the Digital Services Act*, `https://blog.cloudflare.com/digital-services-act`, [Online; accessed 07-August-2024].

[7] P. AG, *Complete guide to GDPR compliance*, `https://gdpr.eu/`, [Online; accessed 07-August-2024].

[8] P. T. Linea, "Linea prover documentation", 2022.

[9] O. Bégassat, A. Belling, T. Chapuis-Chkaiban, and N. Liochon, "A specification for a zk-evm", 2021.

[10] Y. Lindell, *Cryptography and mpc in coinbase wallet as a service (waas)*, 2023.

[11] Privacy and S. Explorations, *Tlsnotary–a mechanism for independently audited https sessions.* `https://github.com/tlsnotary/how_it_works/blob/master/how_it_works.md`, 2014.

[12] F. Zhang, D. Maram, H. Malvai, S. Goldfeder, and A. Juels, "Deco: Liberating web data using decentralized oracles for tls", in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1919–1938.

[13] G. Almashaqbeh and R. Solomon, "Sok: Privacy-preserving computing in the blockchain era", in *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2022, pp. 124–139.

[14] P. Chatzigiannis, K. Chalkias, A. Kate, E. V. Mangipudi, M. Minaei, and M. Mondal, "Sok: Web3 recovery mechanisms", *Cryptology ePrint Archive*, 2023.

[15] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn, "An empirical analysis of anonymity in zcash", in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 463–477.

[16] J. Camenisch, M. Drijvers, and M. Dubovitskaya, “Practical uc-secure delegatable credentials with attributes and their application to blockchain”, in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 683–699.

[17] J. Groth, “On the size of pairing-based non-interactive arguments”, in *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, Springer, 2016, pp. 305–326.

[18] A. Arun, S. Setty, and J. Thaler, “Jolt: Snarks for virtual machines via lookups”, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2024, pp. 3–33.

[19] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The blockchain model of cryptography and privacy-preserving smart contracts”, in *2016 IEEE symposium on security and privacy (SP)*, IEEE, 2016, pp. 839–858.

[20] S. Agrawal, P. Miao, P. Mohassel, and P. Mukherjee, “Pasta: Password-based threshold authentication”, in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2042–2059.

[21] C. Mainka, V. Mladenov, J. Schwenk, and T. Wich, “Sok: Single sign-on security—an evaluation of openid connect”, in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2017, pp. 251–266.

[22] M. Isaac and S. Frenkel, “Facebook security breach exposes accounts of 50 million users”, *The New York Times*, vol. 28, 2018.

[23] S. Wachter, “Normative challenges of identification in the internet of things: Privacy, profiling, discrimination, and the gdpr”, *Computer law & security review*, vol. 34, no. 3, pp. 436–449, 2018.

[24] P. Dunphy and F. A. Petitcolas, “A first look at identity management schemes on the blockchain”, *IEEE Security & Privacy*, vol. 16, no. 4, pp. 20–29, 2018.

[25] G. Wood *et al.*, “Ethereum: A secure decentralised generalised transaction ledger”, *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[26] P. Xia, H. Wang, Z. Yu, X. Liu, X. Luo, and G. Xu, “Ethereum name service: The good, the bad, and the ugly”, *arXiv preprint arXiv:2104.05185*, 2021.

[27] W. Chang, G. Rocco, B. Millegan, N. Johnson, and O. Terbu, *Erc-4361: Sign-in with ethereum [draft]*, `https://eips.ethereum.org/EIPS/eip-4361`, [Online serial] Accessed: 2023-11-15, Oct. 2021.

[28] M. Sporny, D. Longley, and D. Chadwick, “Verifiable credentials data model 1.0”, *W3C, W3C Candidate Recommendation, March*, 2019.

[29] D. Reed, M. Sporny, D. Longley, C. Allen, R. Grant, M. Sabadello, and J. Holt, “Decentralized identifiers (dids) v1. 0”, *Draft Community Group Report*, 2020.

[30] K. N. Chadwick and J. Vercammen, *Openid for verifiable credentials*, 2022.

[31] W. W. W. Consortium, *Verifiable credentials data model v2.0*, `https://www.w3.org/TR/vc-data-model-2.0/`, 2023.

[32] M. Chase, C. Ganesh, and P. Mohassel, "Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials", in *Annual International Cryptology Conference*, Springer, 2016, pp. 499–530.

[33] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system", *Decentralized Business Review*, p. 21 260, 2008.

[34] J. Benet, "Ipfs-content addressed, versioned, p2p file system", *arXiv preprint:1407.3561*, 2014.

[35] F. Vercauteren *et al.*, "Final report on main computational assumptions in cryptography", *European Network of Excellence in Cryptography II*, vol. 11, 2013.

[36] M. Bellare and P. Rogaway, "The security of triple encryption and a framework for code-based game-playing proofs", in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2006, pp. 409–426.

[37] V. Shoup, "Sequences of games: A tool for taming complexity in security proofs", *cryptology eprint archive*, 2004.

[38] J. Benaloh and M. De Mare, "One-way accumulators: A decentralized alternative to digital signatures", in *Workshop on the Theory and Application of Cryptographic Techniques*, Springer, 1993, pp. 274–285.

[39] F. Baldimtsi, J. Camenisch, M. Dubovitskaya, A. Lysyanskaya, L. Reyzin, K. Samelin, and S. Yakoubov, "Accumulators with applications to anonymity-preserving revocation", in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2017, pp. 301–315.

[40] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials", in *Annual international cryptology conference*, Springer, 2002, pp. 61–76.

[41] M. P. Jhanwar and P. R. Tiwari, "Trading accumulation size for witness size: A Merkle tree based universal accumulator via subset differences.", *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 1186, 2019.

[42] J. Lapon, M. Kohlweiss, B. De Decker, and V. Naessens, "Performance analysis of accumulator-based revocation mechanisms", in *IFIP International Information Security Conference*, Springer, 2010, pp. 289–301.

[43] N. Barić and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees", in *International conference on the theory and applications of cryptographic techniques*, Springer, 1997, pp. 480–494.

[44] K. S. McCurley, "The discrete logarithm problem", in *Proc. of Symp. in Applied Math*, USA, vol. 42, 1990, pp. 49–74.

[45] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to iops and stateless blockchains", in *Annual International Cryptology Conference*, Springer, 2019, pp. 561–586.

[46] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes", in *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*, Springer, 1999, pp. 223–238.

[47] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ecdsa with fast trustless setup", in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1179–1194.

[48] T. Chou and C. Orlandi, "The simplest protocol for oblivious transfer", in *Progress in Cryptology–LATINCRYPT 2015: 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings 4*, Springer, 2015, pp. 40–58.

[49] R. Dahlberg, T. Pulls, and R. Peeters, "Efficient sparse merkle trees: Caching strategies and secure (non-) membership proofs", in *Secure IT Systems: 21st Nordic Conference, NordSec 2016, Oulu, Finland, November 2-4, 2016. Proceedings 21*, Springer, 2016, pp. 199–215.

[50] H. Ritzdorf, K. Wüst, A. Gervais, G. Felley, and S. Capkun, "Tls-n: Non-repudiation over tls enabling-ubiquitous content signing for disintermediation", *Cryptology ePrint Archive*, 2017.

[51] A. Nitulescu, *Zk-snarks: A gentle introduction*, 2020.

[52] J. Thaler *et al.*, "Proofs, arguments, and zero-knowledge", *Foundations and Trends® in Privacy and Security*, vol. 4, no. 2–4, pp. 117–660, 2022.

[53] C. Baum, J. H.-y. Chiang, B. David, and T. K. Frederiksen, "Sok: Privacy-enhancing technologies in finance", *Cryptology ePrint Archive*, 2023.

[54] D. Rathee, G. V. Policharla, T. Xie, R. Cottone, and D. Song, "Zebra: Anonymous credentials with practical on-chain verification and applications to kyc in defi", *Cryptology ePrint Archive*, 2022.

[55] M. Rosenberg, J. White, C. Garman, and I. Miers, "Zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure", in *2023 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2023, pp. 790–808.

[56] P. Grubbs, A. Arun, Y. Zhang, J. Bonneau, and M. Walfish, "{Zero-knowledge} middleboxes", in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 4255–4272.

[57] C. Zhang, Z. DeStefano, A. Arun, J. Bonneau, P. Grubbs, and M. Walfish, "Zombie: Middleboxes that {don't} snoop", in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 1917–1936.

[58] J. Lauinger, J. Ernstberger, A. Finkenzeller, and S. Steinhorst, "Janus: Fast privacy-preserving data provenance for tls", *Proceedings on Privacy Enhancing Technologies*, 2025.

[59] C. Reitwiessner, "Zksnarks in a nutshell", *Ethereum blog*, vol. 6, pp. 1–15, 2016.

[60] A. R. Block, A. Garreta, J. Katz, J. Thaler, P. R. Tiwari, and M. Zajac, "Fiat-shamir security of fri and related snarks", *Cryptology ePrint Archive*, 2023.

[61] M. Bellés-Muñoz, M. Isabel, J. L. Muñoz-Tapia, A. Rubio, and J. Baylina, "Circom: A circuit description language for building zero-knowledge applications", *IEEE Transactions on Dependable and Secure Computing*, 2022.

[62] A. Ozdemir, F. Brown, and R. S. Wahby, "Circ: Compiler infrastructure for proof systems, software verification, and more", in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 2248–2266.

[63] D. Benarroch, K. Gurkan, R. Kahat, A. Nicolas, and E. Tromer, "Zkinterface, a standard tool for zero-knowledge interoperability", in *2nd ZKProof Workshop. https://docs. zkproof. org/pages/standards/acceptedworkshop2/proposal–zk-interop-zkinterface. pdf*, 2019.

[64] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge", *Cryptology ePrint Archive*, 2019.

[65] B. Chen, B. Bünz, D. Boneh, and Z. Zhang, "Hyperplonk: Plonk with linear-time prover and high-degree custom gates", in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2023, pp. 499–530.

[66] M. Babel and J. Sedlmeir, "Bringing data minimization to digital wallets at scale with general-purpose zero-knowledge proofs", *arXiv preprint arXiv:2301.00823*, 2023.

[67] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Fast reed-solomon interactive oracle proofs of proximity", in *45th international colloquium on automata, languages, and programming (icalp 2018)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[68] ——, "Scalable, transparent, and post-quantum secure computational integrity", *Cryptology ePrint Archive*, 2018.

[69] D. Boneh, J. Drake, B. Fisch, and A. Gabizon, "Efficient polynomial commitment schemes for multiple points and polynomials", *Cryptology ePrint Archive*, 2020.

[70] M. Jawurek, F. Kerschbaum, and C. Orlandi, "Zero-knowledge using garbled circuits: How to prove non-algebraic statements efficiently", in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 955–966.

[71] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, "Sok: General purpose compilers for secure multi-party computation", in *2019 IEEE symposium on security and privacy (SP)*, IEEE, 2019, pp. 1220–1237.

[72] A. Shamir, "How to share a secret", *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[73] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, "A cryptographic analysis of the tls 1.3 handshake protocol candidates", in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1197–1210.

[74] C. Adams, S. Farrell, T. Kause, and T. Mononen, "Internet x. 509 public key infrastructure certificate management protocol (cmp)", Tech. Rep., 2005.

[75] Y. Lindell, "Secure multiparty computation for privacy preserving data mining", in *Encyclopedia of Data Warehousing and Mining*, IGI global, 2005, pp. 1005–1009.

[76] A. C.-C. Yao, "How to generate and exchange secrets", in *27th annual symposium on foundations of computer science (Sfcs 1986)*, IEEE, 1986, pp. 162–167.

[77] Y. Huang, "Practical secure two-party computation", *dated: Aug*, 2012.

[78] Y. Lindell and B. Pinkas, "Secure two-party computation via cut-and-choose oblivious transfer", *Journal of cryptology*, vol. 25, pp. 680–722, 2012.

[79] X. Wang, S. Ranellucci, and J. Katz, "Authenticated garbling and efficient maliciously secure two-party computation", in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 21–37.

[80] Y. Huang, J. Katz, and D. Evans, "Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution", in *2012 IEEE Symposium on Security and Privacy*, IEEE, 2012, pp. 272–284.

[81] Y. Lindell, *Building mpc wallets – challenges and solutions*, `http://web.archive.org/web/20230312005631/https://cyber.biu.ac.il/wp-content/uploads/2022/11/Building-MPC-Wallets.pdf`, 2022.

[82] J. Lauinger, S. Bezmez, J. Ernstberger, and S. Steinhorst, "Portal: Time-bound and replay-resistant zero-knowledge proofs for single sign-on", in *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, IEEE, 2024, pp. 1–7. DOI: `10.1109/ICBC59979.2024.10634362`.

[83] J. Lauinger, J. Ernstberger, E. Regnath, M. Hamad, and S. Steinhorst, "A-poa: Anonymous proof of authorization for decentralized identity management", in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, IEEE, 2021, pp. 1–9.

[84] J. Ernstberger, J. Lauinger, F. Elsheimy, L. Zhou, S. Steinhorst, R. Canetti, A. Miller, A. Gervais, and D. Song, "Sok: Data sovereignty", in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2023, pp. 122–143. DOI: `10.1109/EuroSP57164.2023.00017`.

[85] Cloudflare, *What is data sovereignty?*, `https://www.cloudflare.com/en-gb/learning/privacy/what-is-data-sovereignty/`, 2024.

[86] C. Garman, M. Green, and I. Miers, "Decentralized anonymous credentials", *Cryptology ePrint Archive*, 2013.

[87] F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin, "Can a public blockchain keep a secret?", in *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part I 18*, Springer, 2020, pp. 260–290.

[88] E. Kokoris Kogias, E. C. Alp, L. Gasser, P. S. Jovanovic, E. Syta, and B. A. Ford, "Calypso: Private data management for decentralized ledgers", *Proceedings of the VLDB Endowment*, vol. 14, no. 4, pp. 586–599, 2021.

[89] A. Narayan, A. Feldman, A. Papadimitriou, and A. Haeberlen, "Verifiable differential privacy", in *Proceedings of the Tenth European Conference on Computer Systems*, 2015, pp. 1–14.

[90] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts", in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2019, pp. 185–200.

[91] W. Li and C. J. Mitchell, "User access privacy in oauth 2.0 and openid connect", in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2020, pp. 664–6732.

[92] P. Labs, *Polygonid: A blockchain-native identity system*, `https://polygonid.com/`, Accessed: 2023-03-04.

[93] S. A. Kakvi, K. M. Martin, C. Putman, and E. A. Quaglia, "Sok: Anonymous credentials", in *International Conference on Research in Security Standardisation*, Springer, 2023, pp. 129–151.

[94] D. Maram, H. Malvai, F. Zhang, N. Jean-Louis, A. Frolov, T. Kell, T. Lobban, C. Moy, A. Juels, and A. Miller, "Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability", in *2021 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2021, pp. 1348–1366.

[95] K. Baghery, M. Kohlweiss, J. Siim, and M. Volkhov, "Another look at extraction and randomization of groth's zk-snark", in *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I 25*, Springer, 2021, pp. 457–475.

[96] B. Edgington, *Upgrading ethereum: 2.9.2 randomness.* `https://eth2book.info/capella/part2/building_blocks/randomness/`, 2023.

[97] J. Ernstberger*, J. Lauinger*, Y. Wu, A. Gervais, and S. Steinhorst, "Origo: Proving provenance of sensitive data with constant communication", *Proceedings on Privacy Enhancing Technologies*, 2025.

[98] G. Botrel, T. Piellard, Y. E. Housni, I. Kubjas, and A. Tabaie, *Consensys/gnark: V0.9.0*, version v0.9.0, Feb. 2023. DOI: `10.5281/zenodo.5819104`. [Online]. Available: `https://doi.org/10.5281/zenodo.5819104`.

[99] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity", in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2016, pp. 191–219.

[100] S. Friebe, I. Sobik, and M. Zitterbart, "Decentid: Decentralized and privacy-preserving identity storage system using smart contracts", in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, IEEE, 2018, pp. 37–42.

[101] F. Baldimtsi, K. K. Chalkias, Y. Ji, J. Lindstrøm, D. Maram, B. Riva, A. Roy, M. Sedaghat, and J. Wang, "Zklogin: Privacy-preserving blockchain authentication with existing credentials", *arXiv preprint arXiv:2401.11735*, 2024.

[102] S. Motepalli, L. Freitas, and B. Livshits, "Sok: Decentralized sequencers for rollups", *arXiv preprint arXiv:2310.03616*, 2023.

[103] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems", in *Conference on the theory and application of cryptographic techniques*, Springer, 1986, pp. 186–194.

[104] J. Li, N. Li, and R. Xue, "Universal accumulators with efficient nonmembership proofs", in *International Conference on Applied Cryptography and Network Security*, Springer, 2007, pp. 253–269.

[105] M. Hölzl, M. Roland, O. Mir, and R. Mayrhofer, "Bridging the gap in privacy-preserving revocation: Practical and scalable revocation of mobile eIDs", in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 2018, pp. 1601–1609.

[106] C. M. Bruhner and O. Linnarsson, *Relay racing with x. 509 mayflies: An analysis of certificate replacements and validity periods in https certificate logs*, 2020.

[107] M. Hölzl, M. Roland, O. Mir, and R. Mayrhofer, "Disposable dynamic accumulators: Toward practical privacy-preserving mobile eids with scalable revocation", *International Journal of Information Security*, pp. 1–17, 2019.

[108] L. Reyzin and S. Yakoubov, "Efficient asynchronous accumulators for distributed PKI", in *International Conference on Security and Cryptography for Networks*, Springer, 2016, pp. 292–309.

[109] J. Lauinger, J. Ernstberger, and S. Steinhorst, "Zkgen: Policy-to-circuit transpiler", in *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, IEEE, 2024, pp. 620–624. DOI: `10.1109/ICBC59979.2024.10634440`.

[110] S. Capkun, E. Ozturk, G. Tsudik, and K. Wüst, "Rosen: Robust and selective non-repudiation (for tls)", in *Proceedings of the 2021 on Cloud Computing Security Workshop*, 2021, pp. 97–109.

[111] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts", in *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security*, 2016, pp. 270–282.

[112] P. Maene, J. Götzfried, R. De Clercq, T. Müller, F. Freiling, and I. Verbauwhede, "Hardware-based trusted computing architectures for isolation and attestation", *IEEE Transactions on Computers*, vol. 67, no. 3, pp. 361–374, 2017.

[113] L. Rosenthol, "C2pa: The world's first industry standard for content provenance", in *Applications of Digital Image Processing XLV*, SPIE, vol. 12226, 2022, 122260P.

[114] S. Longpre, R. Mahari, A. Chen, N. Obeng-Marnu, D. Sileo, W. Brannon, N. Muennighoff, N. Khazam, J. Kabbara, K. Perisetla, *et al.*, "The data provenance initiative: A large scale audit of dataset licensing & attribution in ai", *arXiv preprint arXiv:2310.16787*, 2023.

[115] D. Malkhi. "Exploring proof of solvency and liability verification systems". (2023), [Online]. Available: `https://blog.chain.link/proof-of-solvency/`.

[116] K. Balan, S. Agarwal, S. Jenni, A. Parsons, A. Gilbert, and J. Collomosse, "Ekila: Synthetic media provenance and attribution for generative art", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 913–922.

[117] X. Xie, K. Yang, X. Wang, and Y. Yu, "Lightweight authentication of web data via garble-then-prove", *Cryptology ePrint Archive*, 2023.

[118] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, "A cryptographic analysis of the tls 1.3 handshake protocol", *Journal of Cryptology*, vol. 34, no. 4, pp. 1–69, 2021.

[119] S. Celi, A. Davidson, H. Haddadi, G. Pestana, and J. Rowell, "Distefano: Decentralized infrastructure for sharing trusted encrypted facts and nothing more", *Cryptology ePrint Archive*, 2023.

[120] D. Demmler, G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, and S. Zeitouni, "Automated synthesis of optimized circuits for secure computation", in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1504–1517.

[121] C. Hazay, P. Scholl, and E. Soria-Vazquez, "Low cost constant round mpc combining bmr and oblivious transfer", *Journal of cryptology*, vol. 33, no. 4, pp. 1732–1786, 2020.

[122] P. Grubbs, J. Lu, and T. Ristenpart, "Message franking via committing authenticated encryption", in *Annual International Cryptology Conference*, Springer, 2017, pp. 66–97.

[123] S. Gueron, "Key committing aeads", *Cryptology ePrint Archive*, 2020.

[124] Z. Luo, Y. Jia, Y. Shen, and A. Kate, "Proxying is enough: Security of proxying in tls oracles and aead context unforgeability", *Cryptology ePrint Archive*, 2024.

[125] S. Menda, J. Len, P. Grubbs, and T. Ristenpart, "Context discovery and commitment attacks: How to break ccm, eax, siv, and more", in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2023, pp. 379–407.

[126] Y. Dodis, P. Grubbs, T. Ristenpart, and J. Woodage, "Fast message franking: From invisible salamanders to encryptment", in *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I 38*, Springer, 2018, pp. 155–186.

[127] J. Len, P. Grubbs, and T. Ristenpart, "Partitioning oracle attacks", in *30th USENIX security symposium (USENIX Security 21)*, 2021, pp. 195–212.

[128] N. J. Al Fardan and K. G. Paterson, "Lucky thirteen: Breaking the tls and dtls record protocols", in *2013 IEEE symposium on security and privacy*, IEEE, 2013, pp. 526–540.

[129] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, "Lucky 13 strikes back", in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, 2015, pp. 85–96.

[130] Y. Sheffer, R. Holz, and P. Saint-Andre, *Recommendations for secure use of transport layer security (tls) and datagram transport layer security (dtls)*, 2015.

[131] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications", in *Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II 35*, Springer, 2008, pp. 486–498.

[132] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher", in *2013 IEEE Symposium on Security and Privacy*, IEEE, 2013, pp. 478–492.

[133] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole: Reducing data transfer in garbled circuits using half gates", in *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II 34*, Springer, 2015, pp. 220–250.

[134] G. Botrel, T. Piellard, Y. E. Housni, I. Kubjas, and A. Tabaie, *Consensys/gnark: V0.8.0*, version v0.8.0, Feb. 2023. DOI: `10.5281/zenodo.5819104`. [Online]. Available: `https://doi.org/10.5281/zenodo.5819104`.

[135] H. Xue, M. H. Au, M. Liu, K. Y. Chan, H. Cui, X. Xie, T. H. Yuen, and C. Zhang, "Efficient multiplicative-to-additive function from joye-libert cryptosystem and its application to threshold ecdsa", in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2974–2988.

[136] D. Tymokhanov and O. Shlomovits, "Alpha-rays: Key extraction attacks on threshold ecdsa implementations", *Cryptology ePrint Archive*, 2021.

[137] K. Bhargavan, I. Boureanu, A. Delignat-Lavaud, P.-A. Fouque, and C. Onete, "A formal treatment of accountable proxying over tls", in *2018 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2018, pp. 799–816.

[138] N. Döttling, S. Garg, Y. Ishai, G. Malavolta, T. Mour, and R. Ostrovsky, "Trapdoor hash functions and their applications", in *Annual International Cryptology Conference*, Springer, 2019, pp. 3–32.

[139] S. Takagi, F. Kato, Y. Cao, and M. Yoshikawa, “Asymmetric differential privacy”, in *2022 IEEE International Conference on Big Data (Big Data)*, IEEE, 2022, pp. 1576–1581.

[140] D. Kang, T. Hashimoto, I. Stoica, and Y. Sun, “Zk-img: Attested images via zero-knowledge proofs to fight disinformation”, *arXiv preprint arXiv:2211.04775*, 2022.

[141] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal, “{Raptor}: Routing attacks on privacy in tor”, in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 271–286.

[142] Cloudflare, *Cloudflare radar’s new bgp origin hijack detection system*, `https://blog.cloudflare.com/bgp-hijack-detection/`, 2024.

[143] J. Lu and J. Kim, “Attacking 44 rounds of the shacal-2 block cipher using related-key rectangle cryptanalysis”, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 91, no. 9, pp. 2588–2596, 2008.

[144] Chainlink, *Unlocking web3 identity: Blockchains, credentials, and oracles*, `https://blog.chain.link/web3-identity-blockchains-credentials-oracles/`, Nov. 2022.

[145] a16z crypto, *Zkdocs: Schema to snark circuit transpilation*, `https://github.com/a16z/zkdocs/tree/main/zkdocs-backend`, Dec. 2023.

[146] L. Wang, J. P. Near, N. Somani, P. Gao, A. Low, D. Dao, and D. Song, “Data capsule: A new paradigm for automatic compliance with data privacy regulations”, in *Heterogeneous Data Management, Polystores, and Analytics for Healthcare: VLDB 2019 Workshops, Poly and DMAH, Los Angeles, CA, USA, August 30, 2019, Revised Selected Papers 5*, Springer, 2019, pp. 3–23.

[147] S. Sen, S. Guha, A. Datta, S. K. Rajamani, J. Tsai, and J. M. Wing, “Bootstrapping privacy compliance in big data systems”, in *2014 IEEE Symposium on Security and Privacy*, IEEE, 2014, pp. 327–342.

[148] F. McSherry and K. Talwar, “Mechanism design via differential privacy”, in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS’07)*, IEEE, 2007, pp. 94–103.

[149] S. Garg, C. Gentry, A. Sahai, and B. Waters, “Witness encryption and its applications”, in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, 2013, pp. 467–476.

[150] V. Goyal, A. Kothapalli, E. Masserova, B. Parno, and Y. Song, “Storing and retrieving secrets on a blockchain”, in *IACR International Conference on Public-Key Cryptography*, Springer, 2022, pp. 252–282.

[151] T. Yurek, Z. Xiang, Y. Xia, and A. Miller, “Long live the honey badger: Robust asynchronous {dpss} and its applications”, in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 5413–5430.

[152] S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song, “Churp: Dynamic-committee proactive secret sharing”, in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2369–2386.

[153] C.-P. Schnorr, "Efficient identification and signatures for smart cards", in *Conference on the Theory and Application of Cryptology*, Springer, 1989, pp. 239–252.

[154] E. Fujisaki and T. Okamoto, "Statistical zero knowledge protocols to prove modular polynomial relations", in *Annual International Cryptology Conference*, Springer, 1997, pp. 16–30.

[155] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems", *SIAM Journal on computing*, vol. 18, no. 1, pp. 186–208, 1989.

[156] M. Fischlin, "Trapdoor commitment schemes and their applications.", Ph.D. dissertation, Citeseer, 2001.

[157] A. Sahai and S. Vadhan, "A complete problem for statistical zero knowledge", *Journal of the ACM (JACM)*, vol. 50, no. 2, pp. 196–249, 2003.

[158] T. Okamoto, "On relationships between statistical zero-knowledge proofs", *Journal of Computer and System Sciences*, vol. 60, no. 1, pp. 47–108, 2000.

[159] U. Hohenberger, *Honest verifier zk and fiat-shamir (lecture 1),(2007).*

[160] O. Goldreich, A. Sahai, and S. Vadhan, "Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge", in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 1998, pp. 399–408.

[161] Y. Oren, "On the cunning power of cheating verifiers: Some observations about zero knowledge proofs", in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, IEEE, 1987, pp. 462–471.

[162] R. Canetti, Y. Chen, J. Holmgren, A. Lombardi, G. N. Rothblum, R. D. Rothblum, and D. Wichs, "Fiat-shamir: From practice to theory", in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 2019, pp. 1082–1090.