Proceedings on Privacy Enhancing Technologies ; 2020 (3):327–355

Wouter Lueks*, Brinda Hampiholi, Greg Alpár, and Carmela Troncoso

Tandem: Securing Keys by Using a Central Server While Preserving Privacy

Abstract: Users' devices, e.g., smartphones or laptops, are typically incapable of securely storing and processing cryptographic keys. We present TANDEM, a novel set of protocols for securing cryptographic keys with support from a central server. TANDEM uses *one-time-use key-share tokens* to preserve users' privacy with respect to a malicious central server. Additionally, TANDEM enables users to block their keys if they lose their device, and it enables the server to limit how often an adversary can use an unblocked key. We prove TANDEM's security and privacy properties, apply TANDEM to attribute-based credentials, and implement a TANDEM proof of concept to show that it causes little overhead.

Keywords: privacy, threshold cryptography, anonymity

DOI 10.2478/popets-2020-0055
Received 2019-11-30; revised 2020-03-15; accepted 2020-03-16.

1 Introduction

Privacy-preserving technologies are increasingly deployed in real-life scenarios, such as identity management, e.g., Kryptic[1] and IRMA [4], and digital currencies [60] and products, e.g., the Vega Protocol[2]. These technologies rely on cryptographic protocols and thus their security and privacy properties hinge on the security of the underlying cryptographic keys.

Privacy-preserving cryptographic protocols are typically executed on users' devices such as phones, tablets, and laptops. These devices, however, are notoriously hard to secure and the cryptographic keys are often stored or processed insecurely. A common approach to increase the security of the users' keys is to rely on threshold cryptography to not store the entire key in one device. Instead, these protocols secret-share the user's key between two or more parties, and enable the key's use without ever reconstructing it. This prevents passive and active attackers from learning the key. Typical solutions, such as Shatter [7], assume the user has multiple trustworthy devices that can hold shares. While effective, this approach has availability and usability issues. To address these issues, some approaches instead rely on a highly available central server to store the shares that are not held in the user's device.

Naïvely using threshold cryptography with a central server, however, can harm the users' privacy. As the central server is involved in every key use, it learns the users' key-usage patterns. This information can enable deanonymization of anonymous transactions by correlating key uses with public activities; e.g., by correlating key usages with updates to a blockchain ledger [44, 49].

We present TANDEM, a set of protocols that augment threshold-cryptographic schemes in order to enable the use of a central server as share holder for increased security, while *preserving* the privacy of key-usage patterns. In TANDEM, users obtain *one-time-use key-share tokens* from the central server which contain randomized versions of the central server's key share. To use their key, users send a key-share token to the server via an anonymous communication channel. The embedded randomized key share enables the server to run the threshold-cryptographic protocol *without* learning the user's identity.

The construction of key-share tokens decouples the obtaining and using of tokens. The one-time property provides two security features: the blocking of keys and the rate limiting of unblocked keys. These hold even against active attackers that can compromise the user's device, and thus obtain the user's key share and key-share tokens, and can observe usage of the key.

TANDEM can be used to secure the keys of any cryptographic scheme (e.g., encryption, signature, or payments) for which a linearly randomizable threshold-cryptographic version exists; this version does not require information that identifies the user besides the

***Corresponding Author: Wouter Lueks:** SPRING Lab, EPFL; E-mail: wouter.lueks@epfl.ch
Brinda Hampiholi: Philips Research, all work done while a PhD student at Radboud University; E-mail: brinda.hampiholi@philips.com
Greg Alpár: Open University of the Netherlands, and Radboud University; E-mail: greg.alpar@ou.nl
Carmela Troncoso: SPRING Lab, EPFL; E-mail: carmela.troncoso@epfl.ch

1 https://kryptik.com/
2 https://vegaprotocol.io/

key; and this version is secure when operating on linearly randomized keys. Not all threshold-cryptographic schemes satisfy these properties. For example, Tandem cannot be applied to existing threshold DSA schemes because either they are multiplicative [57] or they require identifying information [40, 41].

To demonstrate the potential of Tandem, we apply it to a threshold version of BBS+ attribute-based credentials (ABCs). ABCs [8, 18, 21, 27] protect users' anonymity during authentication. Introducing a central server that could learn which services users access defeats the very purpose of ABCs. Tandem enables users to securely use their credential keys *without* the central server learning who is using the key, preserving the user's privacy even if the central server and the service provider collude. Tandem can be used to extend many privacy-enhancing technologies such as other attribute-based credential schemes [18, 21, 27], group signatures [14, 28], and electronic cash schemes [23, 60].

Tandem can also be beneficial to non-privacy-preserving cryptographic primitives. For instance, it can be used with threshold variants of Schnorr signatures [42] or ElGamal-based encryption schemes [38, 73] to increase key security without revealing usage patterns to the third party storing the key shares.

We evaluate the practicality of the Tandem protocols on a prototype C implementation. When using the key, Tandem introduces a $50-100$ ms overhead on the central server with respect to traditional threshold-cryptographic solutions. For the user, it only adds 5–25 ms overhead. This is negligible with respect to the delay introduced by the use of anonymous communications, which is needed in privacy-preserving use cases.

In summary, we make the following contributions:

✓ We formalize the security and privacy properties required when using a threshold-cryptographic protocol with a central server.

✓ We introduce Tandem. It enables the use of threshold-cryptographic protocols with a central server without revealing key-usage patterns. Tandem also enables blocking and rate limiting of key usage against active attackers.

✓ We provide a threshold version of BBS+ attribute-based credentials [8], and show how Tandem augments its security while retaining its privacy properties.

✓ We prove the security and privacy of Tandem, and validate its practicality on a prototype implementation. The time-critical computations take less than 100 ms, imposing reasonable overhead on server and users.

2 Related Work

Existing solutions to protect cryptographic keys fall into two categories: *single-party* and *decentralized*. Securing single-party software-based solutions is hard [47, 52, 55, 75]. Secure hardware [37, 58, 70] increases security but it is expensive, is not always available or not accessible to developers [5, 59], is harmful to usability [32], or is not flexible enough to run advanced protocols.

Threshold cryptography [17, 33] strengthens cryptographic protocols by distributing the user's secret key among several parties. There exist a number of threshold encryption and signature schemes [3, 34, 42, 43, 45, 65, 68, 72], with versions that can run in users' personal devices [7]. Other works have tackled more complicated protocols, e.g., to distribute the user's secret key in attribute-based credentials [18], or to make threshold-cryptographic versions of zero-knowledge proofs [51].

Many works propose systems in which the user's secret key is shared between a user's device and a central server [15, 16, 20, 26, 54, 56]. These schemes enable users to block their keys, but do not provide user privacy towards the central server as it is essential that users authenticate themselves to the server to enable the server to find the correct key. Adding anonymous communication or retrieval mechanisms does not resolve this privacy problem. Camenisch et al. [26] ensure privacy to some extent in signature schemes by blinding the message being signed during the threshold protocol with the server. Yet, the server learns when and how often the user uses her signing key. Hence, users are vulnerable to timing attacks [49]. Brands' scheme [18] protects against timing attacks as long as the shareholder cannot store a timed log of operations (e.g., when it is a smart card), but not when keys are shared with an online server.

Tandem is designed to increase the privacy of these threshold-cryptographic solutions. We compare the privacy properties obtained when using Tandem with those in previous proposals in Table 1. We consider three privacy aspects: user's anonymity when running the threshold protocol (i.e., need to authenticate); data hiding (e.g., signed message) in the protocol from the server; and usage pattern hiding to avoid timing attacks. Generic schemes focus on the security of the key and thus provide no privacy. The special-purpose designs only protect data involved in the protocol.

Password-hardening services [25, 39] use decentralization to increase security of authentication servers against brute-force attacks. Similarly to Tandem, these schemes introduce a hardening server to rate limits

Table 1. Comparison of generic, special-purpose and Tandem-augmented threshold-cryptographic protocols (TCPS).

	Generic e.g., [42, 73]	S. Purpose [18, 26]	with Tandem
Anonymous key usage	×	×	✓
Hide protocol data	×	✓	(*)
Hide key-usage patterns	×	×	✓

* Achieved by TANDEM if the underlying TCP does.

or block requests from the main authentication server. However, in the password scenario the hardening server is only accessed by the authentication server. Therefore, there are no privacy concerns and indeed these techniques do not provide any privacy protection.

Single-password authentication schemes [2, 48, 69] hide the user's password from potentially malicious authentication servers. Similar to TANDEM, [2, 48] use a central server to help the user to authenticate to the authentication server. These schemes, however, reconstruct the user's authentication key in the user's device, and thus cannot protect against active attackers.

3 Problem Statement

We consider a scenario where *users* are required to perform cryptographic operations on insecure devices (i.e., without secure hardware) to interact with a *service provider* (SP). To keep their keys safe, users use a *central server* to run threshold-cryptographic protocols (TCPs). The TCP protects the *security* of users' keys against *active adversaries* that can compromise a users device, and that can observe the key being used in protocols. As long as the central server remains honest, security is guaranteed: key usage can be blocked and rate-limited.

Users wish to retain the *privacy* they had with respect to the SP before using the central server with TCPs, even if the central server is malicious and colludes with the SP. We call an execution of the protocol between the user and the central server a *transaction.*

Security and privacy properties. We formalize the desired security and privacy properties of the system.

Property 1 (Key security)**.** *The central server should* prevent unauthorized use of the user's key *even if the user's device is compromised. The user must be able to* block a compromised key *at the server so that the attacker can no longer use it.*

Any solution that recomputes the full user's key on the user's device, e.g., by deriving it from a user-entered passphrase or by interacting with a central server [2, 48], does not satisfy this property: an attacker who compromises the user's device can observe the full key. Thereafter, the attacker can use the key indefinitely, making blocking impossible.

Property 2 (Key-use privacy)**.** *The user must have* privacy of key use in transactions*. The central server must not be able to distinguish between two users performing transactions even if it colludes with the service provider (SP). (Unless the SP could distinguish the users, in which case collusion leads to a trivial and unavoidable privacy breach).*

Property 3 (Key rate-limiting)**.** *Users must be able to* limit the rate of usage of their keys *in a given interval of time called an* epoch*.*

Threat model. We assume the central server is available, and follows the protocols to enforce key security and rate-limits on the users' keys (Property 1 and Property 3) in the face of active attackers. Servers that violate this assumption cannot use the user's key as long as the user's device remains honest.

For privacy, the central server may be malicious, i.e., interested in breaching the privacy of users by trying to learn which keys and services they use (Property 2). It can collude with the SP.

Why naïve centralization does not work. Consider a user that secret-shares her key with the central server. When she needs to run a threshold-cryptographic protocol, she authenticates against the central server and jointly executes the TCP with it. This scheme offers key security (Property 1): The server alone cannot use the user's key and if an attacker compromises the user's device the user can authenticate to the central server and request blocking. It also provides key rate-limiting (Property 3): the central server can easily enforce a limit on the number of times the key is used. However, since the user is identified while using the key, the scheme does not achieve key-use privacy (Property 2).

The lack of key-use privacy has implications when the interactions between the user and the SP are anonymous (e.g., showing an anonymous credential). An SP colluding with the central server can exploit time correlations between the times when the authenticated user interacts with the central server and when the anonymous user interacts with the SP to de-anonymize the user. The anonymity set of the user is reduced to the authenticated users interacting with the central server around the transaction time. This attack has been used in the early days of Tor to identify users and hidden services [1, 61]. As this attack relies solely on time correlation between accesses, it cannot be prevented by making the messages seen by the central server and the SP during the TCP cryptographically unlinkable [18].

Straightforward approaches to prevent time-correlation attacks such as introducing delays and introducing dummy requests are difficult to use in practice. Either operations need to be delayed for a long time, ruling out real-time applications such as showing an anonymous credential or performing a payment; or impose high overhead on the users and the central server, and are difficult to generate [11, 31].

4 Tandem at a Glance

We introduce TANDEM, a set of protocols that wrap threshold-cryptographic schemes to provide privacy when the user's key is shared with a central server (*the* TANDEM *server, TS*). TANDEM ensures key security and key rate-limiting, formalized in Game 7.1, and key-use privacy, formalized in Game 7.3, if the TS colludes with service providers (SPs), and Definition 2, if it does not.

For simplicity, we assume that there is only one TANDEM server (TS). Secret-sharing the key with multiple TANDEM servers would increase security and/or robustness, without detriment to privacy. We explain how to do so in Appendix A.2. We also assume that users can use an anonymous communication channel [35, 66] to communicate with the TS and SPs to protect their privacy at the network layer.

Suppose that Alice uses Schnorr's identification protocol to authenticate herself to her bank. Let p be the group order. To prevent others from accessing her account, Alice protects her secret key x using the threshold-cryptographic version of Schnorr's protocol in Fig. 2. The bank knows Alice's public key h. Alice creates shares x_S and x_U such that $x = x_S + x_U$. She gives x_S to a TANDEM server, keeping x_U on her device. We sketch how TANDEM can wrap the threshold-cryptographic Schnorr protocol so that Alice retains the advantage of using a central server, without revealing her actions to this party. Fig. 1 illustrates the process.

TANDEM uses an additive homormorphic encryption scheme $(\mathbf{E}^+_{pk}, \mathbf{D}^+_{sk})$ [50], and a blind signature scheme [8, 10]. The TS generates key-pairs (pk, sk) and (pk_σ, sk_σ) for these respective schemes and publishes pk and pk_σ.

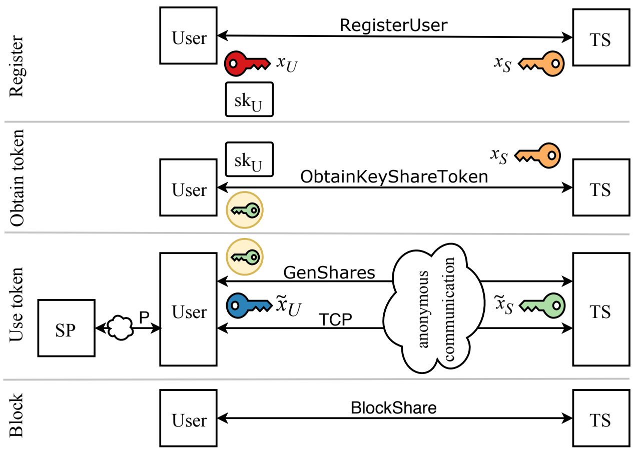


Fig. 1. During registration, the user and TS derive key-shares x_U and x_S of a secret $x = x_U + x_S$. The user stores its authentication key sk_U. Users authenticate before obtaining key-share tokens (🔑), containing a randomized server key share $\tilde{x}_S$. To use her key x anonymously, the user connects to the TS via an anonymous channel and sends a key-share token. The TS recovers the key share $\tilde{x}_S$ and uses it to run the TCP with the user share $\tilde{x}_U$ (the user and SP run protocol P). The user can block her key-share tokens at any time without any key. Inputs are shown above the arrows, outputs below.

Registration. Alice registers with the TS using the RegisterUser protocol. During registration, Alice and the TS jointly compute long-term shares x_U and x_S of a long-term key x. Alice generates a public-private key pair (pk_U, sk_U), and sends pk_U to the TS. The TS sends an homomorphic encryption $\overline{x_S} = \mathbf{E}^+_{pk}(x_S)$ to Alice. Because $\overline{x_S}$ is encrypted against the TS's key, Alice does not learn anything about the share x_S.

Obtain token. Key-share tokens enable Alice to later anonymously use her key (see below). To obtain a token, Alice runs the ObtainKeyShareToken protocol with

Server		**User**		**SP**
$\tilde{x}_S \in \mathbb{Z}_p$		$\tilde{x}_U \in \mathbb{Z}_p$		$h = g^x$
$t_S \in_R \mathbb{Z}_p$		$t_U \in_R \mathbb{Z}_p$		
$u_S = g^{t_S}$	$\xrightarrow{u_S}$	$u_U = g^{t_U}$		
		$u = u_S u_U$	$\xrightarrow{u}$	
	$\xleftarrow{c}$		$\xleftarrow{c}$	$c \in_R \mathbb{Z}_p$
$r_S = t_S + c\tilde{x}_S$	$\xrightarrow{r_S}$	$r_U = t_U + c\tilde{x}_U$		
		$r = r_U + r_S$	$\xrightarrow{r}$	$u \overset{?}{=} g^r h^{-c}$

Fig. 2. A threshold version of Schnorr's proof of identity. The user and the server respectively hold the (fresh) shares $\tilde{x}_U$ and $\tilde{x}_S$ of the private key $x = \tilde{x}_U + \tilde{x}_S$ corresponding to the public key $h = g^x$. They jointly compute a proof of knowledge of x such that $h = g^x$. We write TCP for the threshold-cryptographic protocol between the user and the server, and P for the protocol between the user and the SP.

the TS. First, she uses sk_U to authenticate to the TS. Then, Alice and the TS construct a one-time-use key-share token containing a randomized version of the TS's key share x_S. To do so, Alice picks a large δ and computes $c = \overline{x_S} \cdot \mathbf{E}^+_{pk}(\delta) = \mathbf{E}^+_{pk}(x_S + \delta)$. Then, she sends to the TS a commitment C commiting to both c and her key sk_U. She proves that the committed ciphertext c was constructed by additively randomizing $\overline{x_S}$ and that C contains the correct private key. If the proof is correct, the TS blindly signs the commitment and Alice receives a signature σ on a rerandomized commitment $\tilde{C}$. Alice stores these in her one-time-use key-share token $\tau = (\sigma, \tilde{C}, c, \delta)$. The TS can limit the number of issued tokens, enforcing a rate limit on Alice's key.

Key-share tokens may seem similar to passwords: both unlock functionality. However, unlike passwords, key-share tokens can be verified and used *without* knowing the user's identity. Key-share tokens contain a randomized key share $\tilde{x}_S = \mathbf{D}^+_{sk}(c)$ essential for the TCP. Hence, TANDEM cannot be replaced by a password-hardening service [25, 39]. The randomized key shares contained in the tokens also distinguish them from traditional eCash tokens [23, 30, 60].

Using keys. After obtaining a token $\tau = (\sigma, \tilde{C}, c, \delta)$, Alice can anonymously run threshold-cryptographic protocols with the TS using the GenShares protocol. Alice contacts the TS via an anonymous channel, and sends the signature σ, the commitment $\tilde{C}$, and the randomized encryption c of the TS' key share. She proves that c is committed to in $\tilde{C}$ and that the private key in $\tilde{C}$ does not correspond to a blocked public key. The TS cannot recognize the blindly signed commitment nor the ciphertext c it contains.

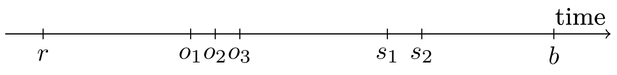
If the signature and proof are correct, the TS derives the fresh share $\tilde{x}_S = \mathbf{D}^+_{sk}(c) \pmod p = x_S + \delta \pmod p$. The size of δ ensures that $\tilde{x}_S$ cannot be linked to the long-term share x_S, thus hiding Alice's identity from the TS. Alice derives her fresh share $\tilde{x}_U = x_U - \delta \pmod p$. By construction, $x = \tilde{x}_S + \tilde{x}_U = x_U + x_S \pmod p$.

Alice can now run the TCP protocol in Fig. 2 to prove her identity to the bank. The TS and Alice use the just computed respective fresh shares $\tilde{x}_S$ and $\tilde{x}_U$. The TS does *not* learn which user ran the TCP protocol. Note that the TS never communicates directly with service providers. Therefore, Alice can use TANDEM without the SP's knowledge.

Blocking keys. Alice can request the TS to block her key by using the BlockShare protocol. She authenticates to the TS using any pre-defined means (e.g., a PUK, or a passphrase; knowledge of the private key sk_U is *not* required). The TS adds Alice's public key pk_U to the list of blocked keys, causing existing tokens to become invalid, and she cannot create new tokens.

Storing the private key sk_U and unused tokens on the user's device is much safer than storing the full key directly. Even if an attacker obtains sk_U and unused tokens, TANDEM guarantees key security and key rate-limiting. Running BlockShare immediately invalidates existing tokens, and prevents the attacker from obtaining new ones. As a result, the attacker can no longer use the user's key x. This is not the case if the attacker can obtain x directly.

Preventing time correlation. To preserve her privacy, the actions of obtaining tokens—where Alice is authenticated—and using tokens—where Alice is anonymous—*must* be uncorrelated, i.e., tokens should not be obtained right before being used. To avoid correlation, Alice can configure her device to obtain tokens at random or regular times (e.g., every night), ensuring that tokens are always available. Suppose Alice obtains fresh tokens every morning, then a time-line of registration (r), obtaining tokens (o_i), using tokens (s_i), and blocking the key (b) events might look as follows:



Note that the obtain and use events do not necessarily follow each other and are *not* correlated. The third token o_3 is unused when Alice blocks her key at time b. This token can thereafter not be used.

5 Cryptographic Preliminaries

Let ℓ be a security parameter. Throughout, $\mathbb{G}$ is a cyclic group of prime order p (of 2ℓ bits) generated by g. We write $\mathbb{Z}_p$ for the integers modulo p; by $[n]$ we denote the set $\{0, \dots, n-1\}$; and by $a \in_R A$ we denote that a is chosen uniformly at random from the set A. We use a cryptographic hash function $H : \{0,1\}^* \to \mathbb{Z}_p$ that maps strings to integers modulo p. For reference, Table 2 in Section 6 summarizes the notation used by TANDEM's building blocks, and Table 3 in Section 6 explains frequently-used symbols in TANDEM.

5.1 Cryptographic Building Blocks

TANDEM relies on a couple of cryptographic building blocks. We use an additive homomorphic encryption scheme given by the algorithms $\mathsf{HE.Keygen}^+$, $\mathbf{E}^+_{pk}, \mathbf{D}^+_{sk}$ with plaintext space $\mathbb{Z}_N$ and space of randomizers $\mathcal{R}$. Let $(pk, sk) = \mathsf{HE.Keygen}^+(1^\ell)$ be a key-pair, then we write $c = \mathbf{E}^+_{pk}(m; \kappa)$ to denote the homomorphic encryption of the message $m \in \mathbb{Z}_N$ using randomness $\kappa \in \mathcal{R}$. The scheme is additively homomorphic, so

$$\mathbf{E}^+_{pk}(m_1; \kappa_1)\mathbf{E}^+_{pk}(m_2; \kappa_2) = \mathbf{E}^+_{pk}(m_1 + m_2 \pmod N); \kappa_1\kappa_2).$$

Our proof of concept uses Joye and Libert's encryption scheme [50], see Appendix A.1, but Paillier's scheme [62] would also work.

TANDEM uses two computationally hiding and binding commitment schemes. First, by $\mathsf{Commit}(m, r)$ we denote a commitment function that takes a message $m \in \mathbb{Z}_p$ and a randomizer $r \in \mathbb{Z}_p$. Analogously, we define $\mathsf{Commit}((m_1, \dots, m_k), r)$ to commit to a tuple of messages. We instantiate this scheme using Pedersen's commitments [64]. Second, we denote by $\Delta = \mathsf{ExtCommit}(m, r)$ with $m \in \{0,1\}^*, r \in \{0,1\}^{2\ell}$ an extractable commitment scheme [71]. That is, in our reductions, we can extract the input m used to create a commitment Δ. For example, the instantiation $\mathsf{ExtCommit}(m, r) = H(m\|r)$ is extractable in the random oracle model for H.

To construct key-share tokens the TS signs them using a blind signature scheme supporting attributes [10] given by the following protocols:

- The signer runs $(pk_\sigma, sk_\sigma) = \mathsf{BSA.KeyGen}(1^\ell, k)$ to setup a system for signatures on k attributes. It obtains a public-private key pair (pk_σ, sk_σ).
- The interactive protocol $\mathsf{BSA.BlindSign}(pk_\sigma, C)$ is run by a user and the signer on input of the signer's public key pk_σ and a Pedersen commitment $C = \mathsf{Commit}((a_1, \dots, a_k), r)$ to the attributes. The signer takes its private key sk_σ as private input, whereas the user takes the attributes $(a_1, \dots, a_k)$ and the randomizer r as private input. At the end of protocol, the user obtains the tuple $(\sigma, \tilde{C}, \tilde{r})$ where σ is a signature on $\tilde{C} = \mathsf{Commit}((a_1, \dots, a_k), \tilde{r})$, a fresh commitment to the attributes. The issuer does not learn the values of the attributes nor the resulting signature σ.
- The verifier calls $\mathsf{BSA.Verify}(pk_\sigma, \sigma, \tilde{C})$ to verify the signature σ on commitment $\tilde{C}$. The algorithm outputs $\top$ if the signature is valid, and $\bot$ otherwise.

We require that the scheme has the blind signing property [10]; and that signatures are unforgeable [10]. For example, the scheme by Baldimtsi and Lysyanskaya [10] satisfies these properties.

5.2 Threshold-Cryptographic Protocols

In this paper, we focus on cryptographic protocols run between a user and a service provider, e.g., showing a credential to an SP or spending an electronic coin. The threshold-cryptographic version of such a protocol splits the user's key x and the user's side of the original protocol in two parts, run by different parties. Each party operates on a secret-share of the user's key. Security of the threshold-cryptographic protocol (TCP) ensures that a large enough subset of shares (two in the case of two parties) are required to complete the protocol.

We focus on TCPs where the user's side of the protocol is distributed between the user and the TS. After registration, the user and the TS hold the shares x_U and x_S of x. After running $\mathsf{GenShares}$ with a new token, the user and TS hold fresh key shares $\tilde{x}_U$ and $\tilde{x}_S$. They then run the TCP protocol, which we denote as:

$$\mathsf{P}(\mathsf{in}_{SP}) \leftrightarrow \mathsf{TCP.U}(\tilde{x}_U, \mathsf{in}_U) \leftrightarrow \mathsf{TCP.TS}(\tilde{x}_S), \qquad (1)$$

where the SP, the user and the TS respectively run the interactive programs P, $\mathsf{TCP.U}$ and $\mathsf{TCP.TS}$. The user mediates all interactions between the service provider and the TS. The user and the SP take extra inputs needed for the execution of the target cryptographic protocol denoted as in_U and in_{SP}. We denote the complete protocol from (1) by $\mathsf{TCP}(\tilde{x}_U, \tilde{x}_S, \mathsf{in}_U, \mathsf{in}_{SP})$.

TANDEM can only enhance the privacy (Property 2) of certain TCPs. We formalize the condition that these TCPs should satisfy. To avoid that the TS can recognize the user based on the shares input to the TCP, we randomize long-term secret shares. Thus, we require that TCPs enhanced with TANDEM function with randomized key shares. In addition, our privacy-friendly GenShares protocol requires this randomization to be linear.

For simplicity, we assume that the user's secret $x \in \mathbb{Z}_p$ for some field $\mathbb{Z}_p$ of prime order p (e.g., corresponding to the group $\mathbb{G}$ defined above). Our constructions, however, can be modified to settings with unknown order arising from RSA assumptions. Formally, we require the TCP to be linearly randomizable:

Definition 1. Let $x_U, x_S \in \mathbb{Z}_p$ be secret shares of the user's secret x. Then, we say that the TCP is *linearly randomizable* if for all μ we have that (1) if $\mathsf{TCP}(x_U, x_S, \mathsf{in}_U, \mathsf{in}_{SP})$ completes successfully, then so does $\mathsf{TCP}(x_U - \mu, x_S + \mu, \mathsf{in}_U, \mathsf{in}_{SP})$, and (2) $x_S + \mu$ is independent from x_S.

The first condition implies that the original secret sharing (x_U, x_S) and the randomized secret sharing $(x_U -$

$\delta, x_S + \delta)$ must share the same secret, whereas the second implies that the TS cannot recognize the user from the randomized secret share alone.

Security and privacy properties of TCPs. To ensure that a TCP with TANDEM satisfies the security properties (Property 1 and Property 3) we require that the TCP itself is secure. That is, if the TS no longer uses its share x_S to run its part of the TCP, then no malicious user can successfully complete the TCP with the SP. We formalize this in Game 7.2 in Section 7.

To ensure that a TCP with TANDEM satisfies the privacy property (Property 2) we require that the TCP itself is private with respect to the TS (respectively the TS colluding with the SP): If the TS runs its part of the TCP using a randomized key-share as input, then the TS (respectively the TS and the SP) cannot recognize the user. We formalize this in Game 7.4 in Section 7.

6 The Full Tandem Construction

We now introduce the full TANDEM construction.

Setup. The TS sets up the TANDEM system as follows.

Protocol 1. *The Setup$(1^\ell, 1^k)$ protocol is run by the TS, where ℓ and k are security parameters. The TS generates a public-private key-pairs (pk, sk) = HE.Keygen$^+(1^\ell)$ for the homomorphic encryption scheme and (pk_σ, sk_σ) = BSA.KeyGen$(1^\ell, k+3)$ for the blind signature scheme. The TS publishes pk and pk_σ. Finally, the TS keeps track of an initially empty list of revoked public keys rlist.*

Registering users. When a user first registers at the TS, the TS computes a key-share x_S for that user, and sends her an encrypted version $\overline{x_S} = \mathbf{E}^+_{pk}(x_S)$. To ensure that the TS cannot hide an identifier in higher-order bits of x_S that are not randomized by the user in the remainder of the protocol, the TS proves that the plaintext x_S is in the correct range.

Protocol 2. *The RegisterUser protocol is run between a user and the TS, and proceeds as follows.*

1. *The user opens an encrypted channel to the TS and authenticates it.*
2. *The user U and the TS generate secret shares $x_U \in_R \mathbb{Z}_p$ and $x_S \in_R \mathbb{Z}_p$, respectively. The user also generates a public-private key-pair $(pk_U, sk_U) = (g^{sk_U}, sk_U)$ for $sk_U \in_R \mathbb{Z}_p$ that we use to authenticate the user's device and to revoke the user's tokens. The user sends pk_U to the TS.*
3. *The TS picks $\kappa \in_R \mathcal{R}$, computes $\overline{x_S} = \mathbf{E}^+_{pk}(x_S; \kappa)$ and sends $\overline{x_S}$ to the user. Moreover, the TS sends a range proof to the user that $\overline{x_S}$ is constructed correctly, i.e., that*
$$\mathbf{D}^+_{sk}(\overline{x_S}) \in [0, p). \tag{2}$$
See Appendix E for how to instantiate this proof.
4. *The TS records $(x_S, \overline{x_S}, pk_U)$ for this user, and marks this user as active. The user stores $(x_U, \overline{x_S}, sk_U)$ on her device.*

Table 2. Notation and cryptographic building blocks.

Symbol	Interpretation
$[n]$	The set $\{0, \dots, n-1\}$
ℓ	The security parameter
$\mathbb{G}, g, p$	Cyclic group $\mathbb{G} = \langle g \rangle$ of order p
Additively homomorphic encryption scheme	
HE.Keygen$^+(1^\ell)$	Generate public-private key-pair
$\mathbf{E}^+_{pk}(m; r)$	Encrypt $m \in \mathbb{Z}_N$ with randomizer $r \in \mathcal{R}$
$\mathbf{D}^+_{sk}(c)$	Decrypt ciphertext c
N	Size of additive plaintext domain
$\mathcal{R}$	Space of randomizers
Commitment schemes and hash function	
Commit(m, r)	Commit to $m \in \mathbb{Z}_p$ (or a tuple of messages) with randomizer $r \in \mathbb{Z}_p$
ExtCommit(m, r)	Commit to $m \in \{0,1\}^*$ with randomizer $r \in \{0,1\}^{2\ell}$
$H(s)$	Hash function from $s \in \{0,1\}^*$ to $\mathbb{Z}_p$
Blind signature scheme with attributes	
BSA.KeyGen$(1^\ell, \lambda)$	Generate signer's key-pair for signatures on λ attributes
BSA.BlindSign(pk_σ, C)	Protocol to blindly sign attributes in C
BSA.Verify$(pk_\sigma, \sigma, \tilde{C})$	Verify signature σ on $\tilde{C}$.

In Appendix A.2 we explain how users can use multiple TSs to increase robustness and/or security.

Obtain a key-share token. First, the user randomizes the ciphertext $\overline{x_S}$. However, it seems difficult to prove directly, for example in zero-knowledge, that the randomized ciphertext produced by the user is of the correct form. Therefore, we use a standard cut-and-choose approach [19, 30] to allow the TS to check that the key share it uses in the TCP is a randomization of the correct secret key with overwhelming probability.

Let ℓ_μ be a security parameter. The user constructs $2k$ witness ciphertexts $c_i = \overline{x_S} \cdot \mathbf{E}^+_{pk}(\mu_i; \kappa_i)$ with $\mu_i \in_R [0, 2^{\ell_\mu})$ and $\kappa_i \in_R \mathcal{R}$. The users sends commitments C_i to these ciphertexts to the TS. The TS then asks the user to open a subset $\mathcal{D}$ of cardinality k, so

Table 3. Frequently used symbols in TANDEM protocols

Symbol	Interpretation
$\mathcal{D}$	Disclose subset in cut-and-choose construction
δ, μ_i	Randomizers of key shares
k	Token security parameter
ℓ_μ	Length of randomizers μ_i in bits
x	Long-term secret key for a user
pk, sk	Public-private encryption key-pair of TS
pk_σ, sk_σ	Public-private signing key-pair of TS
pk_U, sk_U	Public-private key-pair of the user U
p	Order of the group $\mathbb{G}$
x_U, x_S	Long-term key share held by user resp. TS
$\overline{x_S}$	Homomorphic encryption of x_S
$\tilde{x}_U, \tilde{x}_S$	User's resp. TS' key share output by GenShares
ϵ	The current epoch
σ	Blind signature of the TS

that the TS can verify that these k ciphertexts were correctly formed. The user then picks $\delta \in_R [2^{\ell_\mu}, 2^{\ell_\mu+1})$ and $\kappa \in_R \mathcal{R}$ and constructs the randomized ciphertext $c = \overline{x_S} \cdot \mathbf{E}^+_{pk}(\delta; \kappa)$. The choice of δ ensures that it is always bigger than the μ_is. The user constructs a commitment to her private key sk_U, the current epoch ϵ, c, and the remaining k unopened witness ciphertexts c_i. The TS blindly signs this commitment.

We set $\ell_\mu = \lceil \log p \rceil + \ell + \log(k+1) + 2$ to ensure that the $k+1$ plaintexts $x_S + \mu_i$ and $x_S + \delta$ corresponding to the ciphertexts in the token statistically hide x_S. We require that the size of plaintext space N of $\mathbf{E}^+_{pk}$ is bigger than $2^{\ell_\mu+2}$ to ensure no overflows occur.

Protocol 3. *The ObtainKeyShareToken protocol is run between a user and the TS.*

1. *The user opens an encrypted channel to the TS and authenticates it.*
2. *The user recovers $(x_U, \overline{x_S}, sk_U)$ from storage, and authenticates to the TS using sk_U. The TS looks up the corresponding user's record $(x_S, \overline{x_S}, pk_U)$ and aborts if this user exceeded the rate-limit for the current epoch, was banned, or was blocked.*
3. *The TS picks a random subset $\mathcal{D} \subset \{1, \ldots, 2k\}$ of cardinality k of indices of ciphertexts it will check at step 5; and commits to $\mathcal{D}$ by picking $\theta \in_R \{0,1\}^{2\ell}$ and sending $\Delta = \mathit{ExtCommit}(\mathcal{D}, \theta)$ to the user.*
4. *The user picks randomizers $\mu_1, \ldots, \mu_{2k} \in \{0,1\}^{\ell_\mu}$ and $\kappa_1, \ldots, \kappa_{2k} \in \mathcal{R}$ to create witness ciphertexts; and randomizers $r_1, \ldots, r_{2k} \in \mathbb{Z}_p$ and $\xi_1, \ldots, \xi_{2k} \in \{0,1\}^{2\ell}$ for the commitments and sets:*

$$
\begin{aligned}
c_i &= \overline{x_S} \cdot \mathbf{E}^+_{pk}(\mu_i; \kappa_i) \\
C_i &= \mathit{Commit}(H(c_i), r_i) \\
\Delta_i &= \mathit{ExtCommit}((\mu_i, \kappa_i), \xi_i),
\end{aligned} \tag{3}
$$

   for $i = 1, \ldots, 2k$. She sends the commitments $C_1, \ldots, C_{2k}$ and $\Delta_1, \ldots, \Delta_{2k}$ to the TS. Note that the commitments C_i and Δ_i are computationally binding and hiding. We use the extractable commitments Δ_i to extract the inputs μ_i, κ_i in the proofs.
5. *The TS opens the commitment Δ by sending the subset $\mathcal{D}$ and the randomizer θ to the user. The user checks that $\Delta = \mathit{ExtCommit}(\mathcal{D}, \theta)$, and aborts if the check fails.*
6. *The user sends $(c_i, \mu_i, \kappa_i, r_i, \xi_i)_{i \in \mathcal{D}}$ to the TS to open the requested commitments. The TS checks that these values satisfy equation (3) and that $\mu_i < 2^{\ell_\mu}$. If any check fails, the TS bans the user.*
7. *Next, the user creates the randomized ciphertext $c = \overline{x_S} \cdot \mathbf{E}^+_{pk}(\delta; \kappa)$ for $\delta \in_R [2^{\ell_\mu}, 2^{\ell_\mu+1})$ and $\kappa \in_R \mathcal{R}$. Let $\mathcal{H} = \{i_1, \ldots, i_k\} = \{1, \ldots, 2k\} \setminus \mathcal{D}$ be the set of indices of unopened commitments. The user picks $r \in_R \mathbb{Z}_p$ and sends to the TS the commitment*

$$C = \mathit{Commit}((sk_U, \epsilon, H(c), H(c_{i_1}), \ldots, H(c_{i_k})), r)$$

   to her private key sk_U, the epoch ϵ, the ciphertext c, and the unopened witness ciphertexts. Finally, letting $\eta = H(c)$ and $\eta_i = H(c_i)$, she proves in zero-knowledge to the TS that commitment C is correct:

$$
\begin{aligned}
&\mathit{PK}\{((\eta_i, r_i)_{i \in \mathcal{H}}, sk_U, \eta, r) : \\
&\quad \forall i \in \mathcal{H}\, [C_i = \mathit{Commit}(\eta_i, r_i)] \wedge pk_U = g^{sk_U} \wedge \\
&\qquad C = \mathit{Commit}((sk_U, \epsilon, \eta, \eta_{i_1}, \ldots, \eta_{i_k}), r)\}.
\end{aligned}
$$

   The TS checks this proof.
8. *If any check fails, the TS bans the user and aborts the protocol. If all checks pass, the TS runs BSA.BlindSign(pk_σ, C) with the user. The TS takes as private input its signing key sk_σ, the user takes as private input the attributes, and r. Finally, the user obtains the tuple $(\sigma, \tilde{C}, \tilde{r})$ where σ is a blind signature on the commitment $\tilde{C} = \mathit{Commit}((sk_U, \epsilon, H(c), H(c_{i_1}), \ldots, H(c_{i_k}), \tilde{r})$. The user stores the key-share token $\tau = (\sigma, \tilde{C}, \tilde{r}, \epsilon, c, \delta, \kappa, (c_i, \kappa_i, \mu_i)_{i \in \mathcal{H}})$.*

The following lemma states that even if a user is malicious, at least one of the witness ciphertexts c_i must be correctly formed. (See Appendix D for the proof.)

Lemma 1. *Consider a token $\tau = (\sigma, \tilde{C}, \tilde{r}, \epsilon, c, \delta, \kappa, (c_i, \kappa_i, \mu_i)_{i=1,\dots,k})$ obtained using the above protocol by a (potentially malicious) user with corresponding encrypted TS key-share $\overline{x_S}$. Let $\Delta_1, \dots, \Delta_k$ be the set of corresponding commitments used during the obtain step. Then, with probability $1 - 1/\binom{2k}{k}$ there exists an index i^*; and randomizers $\mu^* < 2^{\ell_\mu}$, κ^*, and ξ^* such that:*

$$c_{i^*} = \overline{x_S} \cdot \mathbf{E}^{+}_{pk}(\mu^*; \kappa^*)$$
$$\Delta_{i^*} = \mathsf{ExtCommit}((\mu^*, \kappa^*), \xi^*).$$

Using a homomorphic-CCA secure [67] scheme with targeted malleability that allows adding known randomizers only would obviate the need for extractable commitments. Unfortunately, to the best of our knowledge no such schemes exists. The RCCA scheme by Canetti et al. [29] is not homomorphic, the schemes by Prabhakaran and Rosulek [67] are multiplicatively homomorphic, and the fully homomorphic scheme by Lai et al. [53] is not homomorphic-CCA.

Using a key-share token. When using a token $\tau = (\sigma, \tilde{C}, \tilde{r}, \epsilon, c, \delta, \kappa, (c_i, \kappa_i, \mu_i)_{i=1,\dots,k})$, the user sends $\sigma, \tilde{C}, \epsilon, c, c_1, \dots, c_k$ to the TS, and proves that $\tilde{C}$ contains $\epsilon, H(c), H(c_1), \dots, H(c_k)$ and that the user's tokens have not been revoked. The TS decrypts c and uses the plaintext as the key in the threshold-cryptographic protocol. But how does the TS check if c is correctly formed? To this end, the user reveals the differences $\gamma_i = \delta - \mu_i$ for all $i = 1, \dots, k$. We know from Lemma 1 that at least one index i^* exists such that c_{i^*} is correctly formed. Therefore, if the differences γ_i are correct, then because c_{i^*} is a randomization of x_S, so must be c. In this, key-share tokens differ from Chaum et al.'s e-cash tokens [30], where it suffices that the correct index i^* exists.

Protocol 4. *The* $\mathsf{GenShares}$ *protocol is run between an anonymous user and the TS.*

1. *The user takes $(x_U, \overline{x_S}, sk_U)$ and a token $\tau = (\sigma, \tilde{C}, \tilde{r}, \epsilon, c, \delta, \kappa, (c_i, \kappa_i, \mu_i)_{i=1,\dots,k})$ as input and connects to the TS via an anonymous encrypted channel and authenticates the TS.*
2. *First, the user retrieves the current revocation list* rlist *from the TS. If her public key $pk_U = g^{sk_U}$ is contained in* rlist*, her tokens are blocked and she aborts the protocol, destroys her tokens, and does not use the TS again. Otherwise, she sends $\sigma, \tilde{C}, \epsilon, c, c_1, \dots, c_k$ to the TS together with a zero-knowledge proof that $\tilde{C}$ commits to these values and that her user's tokens have not been revoked:*

   $$\mathsf{PK}\{(sk_U, \tilde{r}) \ : \ g^{sk_U} \notin \mathsf{rlist} \wedge$$
   $$\tilde{C} = \mathsf{Commit}((sk_U, \epsilon, H(c), H(c_1), \dots, H(c_k)), \tilde{r})\}.$$

   The TS verifies the proof; that the signature is valid, i.e., $\mathsf{BSA.Verify}(pk_\sigma, \sigma, \tilde{C}) = \top$*; that it has not seen the signature σ before; and that the epoch ϵ corresponds to the current epoch. The TS aborts if any check fails. The revocation mechanism can be implemented using BLAC (blacklistable anonymous credentials) [74] or dynamic accumulators [24].*
3. *Next, the user computes $\gamma_i = \delta - \mu_i > 0$ and $\nu_i = \kappa_i^{-1} \cdot \kappa$ such that*

   $$c = c_i \cdot \mathbf{E}^{+}_{pk}(\gamma_i; \nu_i) \tag{4}$$

   for $i = 1, \dots, k$. She sends $\gamma_1, \dots, \gamma_k, \nu_1, \dots, \nu_k$ to the TS.
4. *The TS verifies that the γ_is and ν_is satisfy equation (4) and that $0 < \gamma_i < 2^{\ell_\mu + 1}$. The TS aborts if any check fails.*
5. *The TS decrypts c, and sets $\tilde{x}_S = \mathbf{D}^{+}_{sk}(c) \pmod p$.*
6. *The user calculates her key share $\tilde{x}_U$ as:*

   $$\tilde{x}_U \equiv x_U - \delta \pmod p$$

Using Lemma 1, we can show that the decrypted ciphertext c must also be of the right form. (See Appendix D for the proof.)

Lemma 2. *If the tuple $(\epsilon, c, c_1, \dots, c_k,)$ with $\gamma_1, \dots, \gamma_k$ and $\nu_1, \dots, \nu_k$ satisfies equation (4), then with probability $1 - 1/\binom{2k}{k}$ there exists $\delta < 2^{\ell_\mu + 2}$ such that*

$$\mathbf{D}^{+}_{sk}(c) = x_S + \delta$$

where x_S is the TS key-share for the corresponding user.

The range proof in registration is essential. The range proof in equation (2) in the $\mathsf{RegisterUser}$ protocol ensures that the plaintext $x_S = \mathbf{D}^{+}_{sk}(\overline{x_S})$ is small compared to the randomizers μ_i and δ. As a result, the randomized ciphertexts c_i statistically hide x_S. It is not sufficient to skip the range proof and instead choose the randomizers μ_i and δ from the full plaintext domain $[N]$ to hide x_S. Without the range proof, the TS can construct tokens that it can later recognize by exploiting the fact that a large x_S results in a reduction modulo N. More precisely, the TS can set x_S of its target user somewhat large, so that $x_S + \mu_j > N$ (with a non-negligible probability). The user believes that the TS derives $x_S + \mu_j \pmod p$ (because she believes no modular reduction took place) and compensates accordingly. However, the TS actually derives $\tilde{x}_S = (x_S + \mu_j \mod N) \pmod p = x_S + \mu_j - (N \bmod p)$. To test if the current token is from its target user, the TS adds $(N \mod p)$ to $\tilde{x}_S$. If the guess was correct, the TCP completes correctly, otherwise the protocol fails. This allows the TS to detect specific users.

Blocking the Key. To block her key, the user runs the BlockShare protocol with the TS ensuring that no new key-share tokens are created for her, and that all her unspent tokens are blocked. Note that the user does not require knowledge of sk_U, she only needs some mechanism to authenticate to the TS.

Protocol 5. *The user authenticates to the TS. The TS looks up the user's record $(x_S, \overline{x_S}, pk_U)$; marks the user as blocked; and adds pk_U to the revocation list rlist so that the user's unused tokens will be refused.*

We assume the TS is honest with respect to blocking, i.e., it correctly blocks all unspent tokens. A malicious TS could try to attack privacy by revoking the tokens of some honest user. Thereby revealing whether the current user is in the revoked set or not. However, this attack will only work once. As per the first step of the GenShares protocol, the revoked user will detect this revocation and then refuses to use the TS again. Making the list rlist append-only ensures that the TS will always be caught when it maliciously revokes users.

6.1 Alternative Constructions

An alternative method to construct tokens could be to use an authenticated encryption scheme that the user and the TS evaluate using secure multi-party computation [77]. The server inputs its key share x_S while the user inputs the randomizer μ. The user's output is the authenticated encryption of $x_S + \mu$ for the TS's symmetric key which serves as token. To ensure that the TS cannot recognize this token, the protocol should resist malicious servers and the circuit should validate the TS's inputs (i.e., that the encryption key is the same for all users). Similarly, the protocol should resist malicious users to ensure the server's key share does not leak to the user. Such a circuit requires at least 4 block cipher operations and a hash computation. Taking results from recent maliciously secure two-party computation protocols [76] shows that this MPC is requires 1 to 2 orders of magnitude more computation and 2 to 3 orders of magnitude more bandwidth than our custom protocol.

Another simple alternative construction is to let users retrieve $\overline{x_S} = \mathsf{Enc}(x_S)$ using private information retrieval (PIR) via an anonymous channel—the user must still hide her identity. Then, users randomize $\overline{x_S}$ similarly to our construction, and the TS decrypts the ciphertext to recover $\overline{x_S} + \mu$, which it then uses in the TCP. To enable blocking of keys, the TS needs to frequently refresh its encryption keys, effectively invalidating previously retrieved ciphertexts $\overline{x_S}$. This simple protocol, however, has serious drawbacks. First, blocking is only enforced upon key refreshing, thus the time span when compromised keys can be used depends on the refreshing schedule of the TS. Second, because the encryption of x_S for the current period can be randomized as often as the user wants (and the use of PIR precludes record-keeping), this scheme cannot provide rate-limiting. Third, because the TS acts as a decryption oracle for a homomorphic encryption scheme, which is only CPA secure, proving security in this setting requires very strong and non-standard assumptions.

7 Security and Privacy of Tandem

In this section we formalize the security and privacy properties offered by TANDEM. We refer to the appendix for the complete security and privacy proofs.

7.1 Security of Tandem

We capture the security of TANDEM using a security game. It models that if the user's key is compromised (e.g., her device is stolen), the user can block the use of her key, assuming the honesty of the TANDEM server.

Game 7.1. *The* TANDEM security game *is between a challenger controlling the TS and the SP, and an adversary controlling up to n users. The adversary aims to complete the TCP for a blocked or rate-limited user.*
Setup phase *The challenger sets up the TS by running Setup. The challenger also sets up the SP. The challenger runs RegisterUser with the adversary for each of the n users the adversary controls.*
Query phase *During the query phase, the adversary can ask the TS to run the ObtainKeyShareToken and BlockShare protocols with users controlled by the adversary. Moreover, the adversary can make RunTCP queries to the challenger. In response, the TS first runs*

the GenShares protocol with the user (controlled by the adversary), followed by a run of the TCP protocol.

Selection phase *At some point the adversary outputs the identifier of a blocked or rate-limited user U^* on which it wants to be challenged later. The challenger runs BlockShare for user U^* to ensure the user is blocked respectively that the rate-limited user used all tokens.*

Second query phase *The adversary can keep asking the TS to run the ObtainKeyShareToken and BlockShare protocols. The adversary can also make RunTCP queries as before.*

Challenge phase *Finally, upon request of the adversary, the challenger acts as SP in the TCP protocol. At the same time, the adversary may still make queries and run protocols as before. The adversary wins if it successfully completes the TCP with the SP on behalf of the blocked user U^*. To prevent trivial wins, this TCP protocol must be completable only by user U^* (See Appendix B.2 for how to model this for attribute-based credentials).*

In this game, all users are automatically corrupted right from the moment they start the registration protocol. This models the notion that users can even be blocked if an active adversary is present right from the start, and also implies that honest users—which are only corrupted later by an active adversary—can still be blocked.

Of course, to have security using Tandem, the TCP itself must be secure. Hence, we require that even if a malicious user has interacted many times with the TS, she cannot use her key when she does not have access to the TS. We formalize this using the following game.

Game 7.2. *The* TCP security game *is between a challenger controlling the TS and the SP, and the adversary controlling a malicious user.*

Setup phase *During the setup phase, the adversary generates $x_U \in_R \mathbb{Z}_p$, whereas the TS, controlled by the challenger, generates $x_S \in_R \mathbb{Z}_p$.*

Query phase *In the query phase, the adversary can make $\mathsf{TCP}(\delta)$ queries to request that the TS runs $\mathsf{TCP.TS}(x_S + \delta)$ with the user. The adversary is responsible for running TCP.U. Optionally, the adversary-controlled user can communicate with the challenger-controlled SP running $\mathsf{P}()$ as well.*

Challenge phase *In the challenge phase, the adversary is not allowed to make TCP queries. Instead, it interacts solely with the challenger-controlled SP running $\mathsf{P}()$. The adversary wins if the SP accepts.*

Theorem 1. *No PPT adversary can win the* Tandem security game *with non-negligible probability, provided that the TCP is secure (i.e., no PPT adversary can win the TCP security game), the homomorphic encryption scheme is CPA secure, the blind-signature scheme is unforgeable, and the commitment scheme $\mathsf{ExtCommit}(\cdot,\cdot)$ is extractable.*

See Appendix F for the proof.

7.2 Privacy of Tandem

The following game models that Tandem provides key-use privacy: A malicious Tandem server cannot distinguish between two unblocked honest users performing a transaction using the TS even if it colludes with the service provider, provided that the service provider alone cannot distinguish transactions by these users. Thus, users are unlinkable when using their keys at the TS.

Game 7.3. *The* Tandem privacy game with colluding SP *is between a challenger, who controls two honest users U_0 and U_1, and an adversary $\mathcal{A}$ who controls the TS and the SP.*

Setup phase *The adversary $\mathcal{A}$ outputs the number of key-share tokens n_0, n_1 each respective honest user should obtain. The adversary is responsible for setting up the SP and the TS, i.e., it should publish public keys pk and pk_σ. Next, the honest users U_0 and U_1 run RegisterUser with the adversary-controlled TS and then obtain n_0 and n_1 key-share tokens respectively. First, U_0 runs ObtainKeyShareToken n_0 times to obtain tokens $\tau_{0,1}, \ldots, \tau_{0,n_0}$. Then, U_1 runs ObtainKeyShareToken n_1 times to obtain tokens $\tau_{1,1}, \ldots, \tau_{1,n_1}$.*

Query phase *During the query phase, the adversary can make $\mathsf{RunTCP}(U_i, j, \mathsf{in}_U)$ queries to request that user U_i uses token $\tau_{i,j}$ and then runs the TCP with input in_U. If $i \in \{0,1\}$ and user U_i did not use token $\tau_{i,j}$ before, then user U_i, controlled by the challenger, first runs GenShares with the TS using token $\tau_{i,j}$ and then runs $\mathsf{TCP.U}(\mathsf{in}_U)$ with the TS and the SP (running TCP.TS and P respectively).*

Challenge phase *At some point, the adversary outputs a pair of token indices (i_0, i_1) for user U_0 and U_1 respectively on which it wants to be challenged. Let $\tau_0 = \tau_{0,i_0}$ and $\tau_1 = \tau_{1,i_1}$ be the corresponding tokens. The adversary loses if either token τ_0 or τ_1 has been used before or if user U_0 or U_1 detects it is blocked when running GenShares. Then, the challenger picks a bit $b \in \{0,1\}$*

and proceeds as if the adversary made a $\mathsf{RunTCP}(U_b, \tau_b)$ *query followed by a* $\mathsf{RunTCP}(U_{1-b}, \tau_{1-b})$ *query.*
Guess phase *The adversary outputs a guess* b' *of* b*. The adversary wins if* $b' = b$*.*

The privacy game models the fact that there is *no* time correlation between when tokens are obtained by a user, and when they are spent by a user. At the same time, the adversary has full control over the TS and the SP, so this game also models the fact that the TS and the SP *can* correlate events that they see.

Since the SP is controlled by the adversary, the TCP must ensure privacy with respect to the SP and the TS, if all that the TS sees are randomized secret shares. We formalize this in the following game.

Game 7.4. *The* TCP privacy game with colluding SP *is between a challenger controlling honest users* U_0 *and* U_1 *and an adversary* $\mathcal{A}$*, controlling the TS and the SP.*
Setup *The adversary publishes the TS public key and is responsible for setting up the SP. The challenger sets up its users. First, user* U_0 *generates* $x_{0,U} \in_R \mathbb{Z}_p$ *while the TS generates* $x_{0,S} \in_R \mathbb{Z}_p$*, then* U_1 *and TS similarly generate* $x_{1,U}$ *and* $x_{1,S}$*. Finally, the TS sends* $x_{0,S}$ *and* $x_{1,S}$ *to users* U_0 *and* U_1 *respectively.*
Queries *Adversary* $\mathcal{A}$ *can make* $\mathsf{RunTCP}(i, \mathsf{in}_U)$ *queries, to request* U_i *to run the TCP protocol using input* in_U *with the TS and the SP (both controlled by* $\mathcal{A}$*). User* U_i *picks* $\mu \in_R \mathbb{Z}_p$ *and sends the randomized secret-share* $\tilde{x}_S = x_{i,S} + \mu \pmod p$ *to the TS. The user sets* $\tilde{x}_U = x_{i,U} - \mu$ *and runs* $\mathsf{TCP.U}(\tilde{x}_U, \mathsf{in}_U)$ *with the TS and the SP running* $\mathsf{TCP.TS}(\tilde{x}_S)$ *and* P *respectively.*
Challenge *Adversary* $\mathcal{A}$ *outputs an input* in_U*. The challenger picks a bit* $b \in_R \{0,1\}$*. Then the challenger acts as if* $\mathcal{A}$ *first made a* $\mathsf{RunTCP}(b, \mathsf{in}_U)$ *query, and then a* $\mathsf{RunTCP}(1-b, \mathsf{in}_U)$ *query.*
Guess $\mathcal{A}$ *outputs a guess* b' *for* b*,* $\mathcal{A}$ *wins if* $b = b'$*.*

TANDEM also provides key-use privacy against the TS alone, even if the SP can identify users. (If the SP can identify users, then so can the TS and the SP together. We exclude this case to prevent a trivial win.) We model this situation as a variant of the previous two games.

Definition 2. The TANDEM *privacy game with honest SP* and the *TCP privacy game with honest SP* are as in Game 7.3 and Game 7.4 above, however, the challenger controls the SP. The adversary can interact with the SP as a normal user.

Theorem 2. *No PPT adversary can win the* TANDEM privacy game with colluding SP *(respectively the* TANDEM privacy game with honest SP*) with probability non-negligibly better than* $1/2$*, provided that the* TCP *is privacy-friendly (i.e., no PPT adversary can win the TCP privacy game with colluding SP respectively the TCP privacy game with honest SP), the commitment scheme* $\mathsf{Commit}(\cdot,\cdot)$ *is computationally hiding, and that the commitment scheme* $\mathsf{ExtCommit}(\cdot,\cdot)$ *is extractable.*

See Appendix G for the proof.

8 Securing Protocols with Tandem

TANDEM enables the use of a central server in a common class of TCPs without incurring a privacy cost with respect to this server. It operates on specific linearly randomizable TCPs (with corresponding group order p) that satisfy the security an privacy properties in Games 7.2 and 7.4. A central server that is willing to run these TCPs can trivially offer them in a privacy-preserving manner by implementing TANDEM's protocols in Section 6. As a result, the central server can no longer distinguish users when it runs a TCP. Of course, it can still distinguish different TCPs and see inputs to the TCPs that are not protected by TANDEM.

Many protocols already have threshold-cryptographic versions that satisfy the necessary properties. For example, threshold variants of Schnorr signatures [42] and ElGamal-based encryption schemes [38, 73] rely on Shamir secret-sharing and are thus linearly randomizable. A reduction to the original security property shows they also satisfy the TCP security property: given oracle access to a signer or decryptor for a fixed private key, we can answer the $\mathsf{TCP}(\delta)$ queries by modifying the original response. The protocols are also TCP private. The server-side protocols for signing and decrypting operate solely on the secret-share and the common input, the message or ciphertext. In Appendix C we show how to protect ElGamal decryption with TANDEM.

TANDEM can enhance threshold-cryptographic versions of electronic cash [23, 60]; group signatures [14, 28]; and attribute-based credentials (ABC) [8, 18, 21, 27]. Only Brands' scheme has a threshold-cryptographic version [18]. For the others, the threshold-cryptographic versions of the zero-knowledge proofs must be created. As an example, we show how to convert the BBS+

ABC scheme [8] into a TANDEM-suitable threshold-cryptographic scheme.

Not all threshold schemes are compatible with TANDEM. Threshold DSA signatures are notoriously complicated. Some schemes use multiplicative secret-sharing (instead of additive) [36, 57], and others require additional information such as public keys or encryption keys that break the TCP privacy property [40, 41]. Similarly, threshold RSA encryption and decryption [72] need the public modulus, and hence do not satisfy the TCP privacy property.

8.1 Use Case: Attribute-Based Credentials

Attribute-based credentials can be conceptualized as digital equivalents to classic identity documents such as passports. The owner of a credential can selectively disclose any subset of attributes to a service provider so that the validity of the disclosed attributes can be verified. In many ABC systems credentials are unlinkable across disclosures, making users anonymous within the set of users having the same disclosed attributes.

Credentials contain the user's secret key to bind credentials to a user, and to ensure that only the owner can use them. TANDEM can be used to strengthen the security of this key to ensures that credentials cannot be abused while preserving users' privacy.

We now show how to apply TANDEM to BBS+ credentials [8] by converting its issuing and showing protocols into threshold-cryptographic alternatives. BBS+ credentials are anonymous credentials built from BBS+ signatures [8]. BBS+ signatures operate in a pairing setting and rely on discrete-logarithm based assumptions. Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group pair, both of prime order p, generated by g and h respectively. The pairing is given by $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where $\mathbb{G}_T$, also of order p, is generated by $\hat{e}(g, h)$. Let l be the number of attributes. In the BBS+ credential scheme, an issuer randomly chooses generators $B, B_0, .., B_l \in_R \mathbb{G}_1$, picks a private key $sk_I \in_R \mathbb{Z}_p$, and computes $w = h^{sk_I}$. The issuer's public key is $pk_I = (w, B, B_0, .., B_l)$.

Obtaining a credential. Attribute-based credentials contain the user's secret key as an attribute. For simplicity, we describe the TANDEM BBS+ issuance and showing protocols below with two attributes: the secret key x and an issuer-determined attribute a_1. To obtain a credential, the user and the TANDEM server run the following TCP version of the issuance protocol with the issuer. Let $\tilde{x}_U$ and $\tilde{x}_S$ be the shares of the user's secret key $x = \tilde{x}_U + \tilde{x}_S$ held by the user and the TS respectively. The user must commit to her secret key x to allow the issuer to blindly sign it. As the user's secret key is shared between the user and the TS, they both have to participate in creating the commitment. First, the user sends B_0 to the TS so that it can compute $B_0^{\tilde{x}_S}$. Then the user and the TS create a commitment $U = B^{s'} B_0^{\tilde{x}_U} B_0^{\tilde{x}_S} = B^{s'} B_0^{x}$ where $s' \in_R \mathbb{Z}_p$. To prove to the issuer that U is well-formed, the user and the TS construct the proof $\mathsf{PK}\{(x, s') : U = B^{s'} B_0^{x}\}$. The construction of this proof is very similar to the threshold version of Schnorr's protocol in Fig. 2. For completeness, we include the full protocol in Fig. 4 in Appendix B.1. If this proof of knowledge verifies, the issuer randomly generates $s'', e \in_R \mathbb{Z}_p$, calculates

$$A = \left(gB^{s''} U B_1^{a_1}\right)^{\frac{1}{e+sk_I}} \quad \in \mathbb{G}_1,$$

and sends the tuple (A, e, s'') to the user. The user calculates $s = s' + s''$ and stores the credential $\sigma = (A, e, s)$.

Showing a credential. After the issuance protocol, the user can show the credential to authenticate to a service provider. We convert the showing protocol into a TCP that uses the TANDEM server.

Using the showing protocol, the user can prove possession of a credential $\sigma = (A, e, s)$ over her key x and a revealed attribute a_1 (x remains hidden) by convincing the service provider the credential is valid, i.e., that

$$\hat{e}(A, h^e w) = \hat{e}(gB^s B_0^x B_1^{a_1}, h). \tag{5}$$

We follow the approach by Au et al. [8] to prove this in zero-knowledge. Let g_1, g_2 be generators in $\mathbb{G}_1$. First, the user creates commitments $C_1 = A g_2^{r_1}$ and $C_2 = g_1^{r_1} g_2^{r_2}$ for $r_1, r_2 \in_R \mathbb{Z}_p$, and sends them to the SP. Finally, she and the TS engage in the following zero-knowledge proof with the SP:

$$\begin{aligned}\mathsf{PK}\Big\{&(r_1, r_2, \alpha_1, \alpha_2, e, x, s) : C_2 = g_1^{r_1} g_2^{r_2} \wedge C_2^e = g_1^{\alpha_1} g_2^{\alpha_2} \wedge \\ &\hat{e}(C_1, w)\hat{e}(C_1, h)^e = \hat{e}(g, h)\hat{e}(B, h)^s \mathbf{\hat{e}(B_0, h)}^x \cdot \\ &\qquad \hat{e}(B_1, h)^{a_1} \hat{e}(g_2, w)^{r_1} \hat{e}(g_2, h)^{\alpha_1}\Big\}\end{aligned}$$

to prove that she indeed posseses the signature over the hidden and the disclosed attributes and that equation (5) is satisfied. In the proof, $\alpha_1 = er_1$ and $\alpha_2 = er_2$. The user can herself generate the proofs for the first two conjuncts. The third conjuct, however, contains the user's secret key x of which the user only has a share. Thus, the user has to contact the TS to construct this part of the proof. This proof is just a proof of representation, as before, albeit a bit more complex. As a result, a construction similar to Fig. 2 and 4 (in the appendix), allows the user and the TS to jointly compute this proof. See Appendix B.1 for the full protocol.

Security and privacy of the TCPs. These TCPs satisfy the TCP security and privacy notions defined in Section 7. The TS computes zero-knowledge proofs of knowing $\tilde{x}_S$. A malicious user learns nothing about $\tilde{x}_S$ (thus nor x_S) as a result of the zero-knowledge property. Hence, the TCP showing and issuance protocols satisfy the TCP security property (see Game 7.2).

For privacy (see Game 7.4), the TS operates on a randomized key $\tilde{x}_S$, so the TS cannot distinguish users if the SP is honest. The indistinguishability of the credential scheme guarantees that the TS cannot distinguish by colluding with the SP either. Thus, the TCP showing protocol is private for honest and colluding SPs.

We refer to Appendix B.2 for the full TCP security and privacy proofs.

Revocation and rate-limiting with Tandem. When applying Tandem to ABC schemes, the TS can rate-limit and block keys, and therefore rate-limit and revoke credentials. Complex custom solutions create revocable [9, 74] and rate-limitable [22] credentials directly. While the end-result is similar, Tandem makes different trust assumptions.

Consider the case where honest users want to protect themselves against compromise of their credentials. The custom solutions rely on the honesty of SPs to enforce rate-limits and revocations on behalf of the user. Tandem instead ensures rate-limiting and blocking as long as the TS, which the user chooses, is honest.

The Tandem approach in this section, however, cannot protect SPs against malicious users as users are not forced to use Tandem. In Appendix B.3 we show a modification that forces all users to use the (same) TS. With this change, the TS can rate-limit and block any misbehaving users on behalf of issuers and service providers, replacing complex ad-hoc cryptographic techniques.

9 Performance Evaluation

We evaluate Tandem's computational and bandwidth cost. We use the ABC instantiation as a case study and compare its performance without key protection, with vanilla threshold-cryptographic version of the ABC protocols, and with Tandem protection.

Tandem consists of four protocols: RegisterUser, ObtainKeyShareToken, GenShares, and BlockShare. We implemented in C the time-critical protocols, ObtainKeyShareToken and GenShares.[3] We used Pedersen commitments [64] as commitment scheme, and BBS+ credentials [8] to construct the blind signature (we use an extra attribute containing a serial number to ensure that the blind signature can be used only once). We use the RELIC cryptographic library to implement them [6].[4] We use a recent implementation [12] of Joye and Libert's additive homomorphic encryption scheme [50], see Appendix A.1. We set the modulus size to 2048 bits and the size of the plaintext space N to 394 bits, such that $N > 2^{\ell_\mu+2}$ for $k < 64$ (recall ℓ_μ is the size of the randomizers, see page 6). With this setting, encrypting a single 394 bits plaintext takes 0.9 ms whereas it takes 24.2 ms to decrypt a ciphertext.

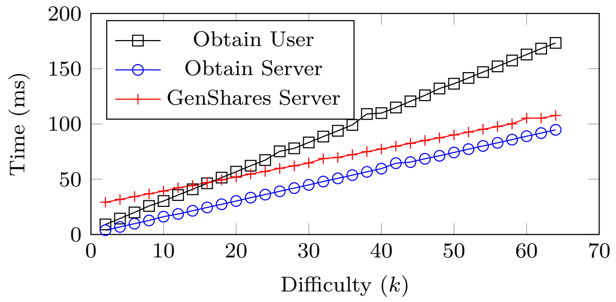


Fig. 3. ObtainKeyShareToken protocol computing time at the user (black) and the server side (blue), and GenShares protocol computation time at the server side (red) for increasing difficulty levels k excluding revocation cost.

We empirically measure performance on a single core of an Intel i7-7700 running at 3.6 GHz.

Obtaining a token. We first justify our choice for the parameters k. Our analysis shows that an attacker can break Tandem's security property by constructing a key-share token for a blocked user with probability $\binom{2k}{k}^{-1}$. Hence, $k = 42$ gives 80 bits of security, and $k = 66$ gives 128 bits security. However, ObtainKeyShareToken is an *interactive* protocol. The success probability of an attacker is limited by how often the TS lets the attacker try to construct a malicious token rather than by the adversary's computational power. As the TS bans users trying to construct malicious tokens, one can choose a smaller k in practice. In a system with 100 000 users, $k = 20$ ensures that the probability that an attacker (corrupting all users) can at least *once* use any blocked key is less than 10^{-6}.

3 Code here: https://github.com/spring-epfl/tandem
4 We use a BLS curve over a 381 bits field in RELIC. This setup ensures 128 bits security, while the group order remains 255 bits.

Fig. 3 shows the computing time (without communication) for the ObtainKeyShareToken protocol at the user (black) and server (blue) for different values of the parameter k. The homomorphic encryption scheme—creating the ciphertexts (user), and checking a subset of these (TS)—dominates the computational cost. Our experiments reveal that the timing variance across executions is negligible. The bandwidth cost is low: users send and receive $1314k + 405$ and $2k + 117$ bytes respectively. For a security level of $k = 20$, the user sends about 26 KiB and receives less than 200 bytes.

Using the key. We first examine the cost of using a key without the token-revocation check. On the user side running GenShares is very cheap: less than 5 ms even for $k = 60$. Users send $594k + 516$ bytes to the server, i.e., 12 KiB for $k = 20$ and 35 KiB for $k = 60$. We show the server's computational cost for recovering the TS key-share from the token in Fig. 3. For a reasonable security level of $k = 20$, the server computational overhead is around 50 ms. The sending of the token in the GenShares protocol can be combined with the request to start the TCP, resulting in no extra latency on top of the delay incurred by the Tor network [35] (1–2 s to send and receive a small amount of data on a fresh circuit[5]). Circuit creation and GenShares can be run preemptively, thereby reducing the user-perceived delay.

TANDEM uses standard revocation techniques to revoke key-share tokens. Therefore, we did not implement revocation. Tokens expire automatically, so we would only need to block tokens from the current epoch. If the number of blocked users per epoch is small (e.g., for short epochs, such as a day), TANDEM could use BLAC [74]. BLAC is simple, but proving that a user's tokens are not revoked has linear complexity. Based on the results by Henry and Goldberg [46], we estimate that for 100 blocked users a user needs an additional 20 ms to prove non-revocation. The TS needs about 10 ms to check this proof. If the number of revoked users per epoch is larger, then it is more efficient to use dynamic accumulators [24], which are more complex, but constant time. We estimate them to have a 10–20 ms cost.

Given the above measurements, a modern 4-core server can participate in approximately 50 TCPs per second (not counting the cost of the application-dependent TCP itself), i.e., serve 3 000 users per minute, requiring about 20 Mbit/s incoming bandwidth.

Table 4. Comparison of computational cost and properties when running the showing protocol of the BBS+ ABC scheme. We compare not using a TCP, using a traditional TCP, and using a TCP with TANDEM ($k = 20$, excluding revocation cost).

	No TCP	Vanilla TCP	TCP + Tandem
Obtain Token			
User	-	-	57 ms
Server	-	-	30 ms
Run Protocol			
User	5 ms	5 ms	5 + 4 ms
Server	-	1 ms	1 + 52 ms
Key blocking	×	✓	✓
Rate limiting	×	✓	✓
Privacy	✓	×	✓

RegisterUser and BlockShare. These protocols are run rarely (only upon registration and for blocking) and are thus not critical for scalability. We estimate the cost for RegisterUser to be well below a second for both the user and the TS (given its similarity with the ObtainKeyShareToken and GenShares protocols, and that the cost of the range proof is around 500 ms). Users have no computational cost when running BlockShare. The server's cost is less than a few milliseconds when using BLAC [46] or dynamic accumulators [24].

Comparison. Table 4 compares the computational cost of creating a single BBS+ showing proof with 5 hidden attributes without key protection, using straightforward TCP version of the disclosure proof, and using the TANDEM-augmented TCP with $k = 20$.

Without a TCP, the credential showing is very fast and, as there is no other party involved in the use of the key, the showing of the credential is perfectly private. However, it is not possible to perform key blocking nor limit the key-usage without changing the credential type (e.g. [22]) *and* trusting the SP. The traditional TCP version, has minimal overhead (1 ms at the server) and provides key blocking and rate limiting, at the cost of privacy. TANDEM provides all three properties. Without taking into account the ObtainKeyShareToken operation that happens offline, the user's overhead is negligible (4 ms), and well below a second (52 ms) for the server. In all cases, the delays due to TANDEM's cryptographic operations are small compared to Tor's network delay.

5 As reported by https://metrics.torproject.org/torperf.html, visited August 31, 2019.

10 Conclusion

Protecting cryptographic keys is imperative to maintain the security of cryptographic protocols. As users' devices are often insecure, the community has turned to threshold-cryptographic protocols to strengthen the security of keys. When run with a central server, however, these protocols raise privacy concerns. In this paper, we have proposed TANDEM, a provably secure scheme that, when composed with threshold-cryptographic protocols, provides privacy-preserving usage of keys. TANDEM also enables users to block their keys and rate-limit their usage. Our prototype implementation shows that for reasonable security parameters TANDEM's protocols run in less than 60 ms, showing TANDEM's practicality.

TANDEM is particularly suited for privacy-friendly applications such as eCash and ABCs because it retains their inherent privacy properties. Yet, TANDEM can be used to strengthen a wide variety of primitives, including signature and encryption schemes, as long as they can be transformed into linearly-randomizable threshold protocols. Using attribute-based credentials we have shown that deriving such a threshold protocol can be done with standard techniques, and that thereafter adding TANDEM is straightforward.

Acknowledgments

This research is partially funded by the NEXTLEAP project[6] within the European Union's Horizon 2020 Framework Program for Research and Innovation (H2020-ICT-2015, ICT-10-2015) under grant agreement 688722; the Netherlands Organization for Scientific Research (NWO); and KPN under project 'Own Your Own Identity (OYOI)'.

References

[1] Timothy G. Abbott, Katherine J. Lai, Michael R. Lieberman, and Eric C. Price. 2007. Browser-Based Attacks on Tor. In *PETS 2007*.

[2] Tolga Acar, Mira Belenkiy, and Alptekin Küpçü. 2013. Single password authentication. *Computer Networks* 57, 13 (2013).

[3] Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. 2006. Simplified Threshold-RSA with Adaptive and Proactive Security. In *EUROCRYPT 2006*.

[4] Gergely Alpár, Fabian van den Broek, Brinda Hampiholi, Bart Jacobs, Wouter Lueks, and Sietse Ringers. 2017. IRMA: Practical, Decentralized and Privacy-friendly Identity Management Using Smartphones. In *HotPETs 2017*.

[5] Android security website. 2017. Developing third party applications with Trusty TEE. https://source.android.com/security/trusty/#third-party_trusty_applications. (2017).

[6] D. F. Aranha and C. P. L. Gouvêa. 2020. RELIC is an Efficient Library for Cryptography. https://github.com/relic-toolkit/relic. (2020).

[7] Erinn Atwater and Urs Hengartner. 2016. Shatter: Using Threshold Cryptography to Protect Single Users with Multiple Devices. In *WISEC 2016*.

[8] Man Ho Au, Willy Susilo, and Yi Mu. 2006. Constant-Size Dynamic *k*-TAA. In *SCN 2006*.

[9] Man Ho Au, Patrick P. Tsang, and Apu Kapadia. 2011. PEREA: Practical TTP-free Revocation of Repeatedly Misbehaving Anonymous Users. *TISSEC* (2011).

[10] Foteini Baldimtsi and Anna Lysyanskaya. 2013. Anonymous credentials light. In *CCS 2013*.

[11] Ero Balsa, Carmela Troncoso, and Claudia Díaz. 2012. OB-PWS: Obfuscation-Based Private Web Search. In *S&P 2012*.

[12] Manuel Barbosa, Dario Catalano, and Dario Fiore. 2017. Labeled Homomorphic Encryption: Scalable and Privacy-Preserving Processing of Outsourced Data. In *ESORICS 2017*.

[13] Mihir Bellare and Shafi Goldwasser. 1997. Verifiable Partial Key Escrow. In *CCS 1997*.

[14] Patrik Bichsel, Jan Camenisch, Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. 2010. Get Shorty via Group Signatures without Encryption. In *SCN 2010*.

[15] Dan Boneh, Xuhua Ding, and Gene Tsudik. 2004. Fine-grained Control of Security Capabilities. *TOIT* (2004).

[16] Dan Boneh, Xuhua Ding, Gene Tsudik, and Chi-Ming Wong. 2001. A Method for Fast Revocation of Public Key Certificates and Security Capabilities. In *USENIX 2001*.

[17] Colin Boyd. 1989. Digital Multisignatures. *Cryptography and Coding* (1989).

[18] Stefan A Brands. 2000. *Rethinking public key infrastructures and digital certificates: building in privacy*. MIT Press.

[19] Gilles Brassard, David Chaum, and Claude Crépeau. 1988. Minimum Disclosure Proofs of Knowledge. *J. Comput. Syst. Sci.* (1988).

[20] Ahto Buldas, Aivo Jürgenson, Aivo Kalu, and Mart Oruaas. 2017. Server-Supported RSA Signatures for Mobile Devices. In *ESORICS 2017*.

[21] Jan Camenisch and Els Van Herreweghen. 2002. Design and Implementation of the *Idemix* Anonymous Credential System. In *CCS 2002*.

[22] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. 2006. How to Win the Clone Wars: Efficient Periodic n-times Anonymous Authentication. In *CCS 2006*.

[23] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2005. Compact E-Cash. In *EUROCRYPT 2005*.

6 https://nextleap.eu

[24] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. 2009. An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials. In *PKC 2009*.

[25] Jan Camenisch, Anja Lehmann, and Gregory Neven. 2015. Optimal Distributed Password Verification. In *CCS 2015*.

[26] Jan Camenisch, Anja Lehmann, Gregory Neven, and Kai Samelin. 2016. Virtual Smart Cards: How to Sign with a Password and a Server. In *SCN 2016*.

[27] Jan Camenisch and Anna Lysyanskaya. 2002. A Signature Scheme with Efficient Protocols. In *SCN 2002*.

[28] Jan Camenisch and Anna Lysyanskaya. 2004. Signature Schemes and Anonymous Credentials from Bilinear Maps. In *CRYPTO 2004*.

[29] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. 2003. Relaxing Chosen-Ciphertext Security. In *CRYPTO 2003*.

[30] David Chaum, Amos Fiat, and Moni Naor. Untraceable Electronic Cash. In *CRYPTO '88*.

[31] Richard Chow and Philippe Golle. 2009. Faking Contextual Data for Fun, Profit, and Privacy. In *WPES 2009*.

[32] Sanchari Das, Andrew Dingman, and L Jean Camp. 2018. Why Johnny Doesn't Use Two Factor A Two-Phase Usability Study of the FIDO U2FSecurity Key. In *FC 2018*.

[33] Yvo Desmedt. 1987. Society and Group Oriented Cryptography: A New Concept. In *CRYPTO '87*.

[34] Yvo Desmedt and Yair Frankel. 1991. Shared Generation of Authenticators and Signatures (Extended Abstract). In *CRYPTO '91*.

[35] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX 2004*.

[36] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. 2018. Secure Two-party Threshold ECDSA from ECDSA Assumptions. In *S&P 2018*.

[37] Jan-Erik Ekberg, Kari Kostiainen, and N. Asokan. 2014. The Untapped Potential of Trusted Execution Environments on Mobile Devices. In *S&P 2014*.

[38] Taher ElGamal. 1984. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO '84*.

[39] Adam Everspaugh, Rahul Chatterjee, Samuel Scott, Ari Juels, and Thomas Ristenpart. 2015. The Pythia PRF Service. In *USENIX 2015*.

[40] Rosario Gennaro and Steven Goldfeder. 2018. Fast Multiparty Threshold ECDSA with Fast Trustless Setup. In *CCS 2018*.

[41] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. 2016. Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security. In *ACNS 2016*.

[42] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *J. Cryptology* (2007).

[43] Rosario Gennaro, Tal Rabin, Stanislaw Jarecki, and Hugo Krawczyk. 2000. Robust and Efficient Sharing of RSA Functions. *J. of Cryptology* (2000).

[44] Steven Goldfeder, Harry A. Kalodner, Dillon Reisman, and Arvind Narayanan. 2018. When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies. *PoPETs* (2018).

[45] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. 2012. Efficient RSA Key Generation and Threshold Paillier in the Two-Party Setting. In *CT-RSA 2012*.

[46] Ryan Henry and Ian Goldberg. 2013. Thinking inside the BLAC box: smarter protocols for faster anonymous blacklisting. In *WPES 2013*. 71–82.

[47] Alex Hern. 2015. Stagefright: new Android vulnerability dubbed 'heartbleed for mobile'. *The Guardian* (2015). https://www.theguardian.com/technology/2015/jul/28/stagefright-android-vulnerability-heartbleed-mobile

[48] Devriş İşler and Alptekin Küpçü. 2017. Threshold Single Password Authentication. In *DPM 2017*.

[49] Husam Al Jawaheri, Mashael Al Sabah, Yazan Boshmaf, and Aiman Erbad. 2018. When A Small Leak Sinks A Great Ship: Deanonymizing Tor Hidden Service Users Through Bitcoin Transactions Analysis. (2018). arXiv:1801.07501

[50] Marc Joye and Benoît Libert. 2013. Efficient Cryptosystems from 2^k-th Power Residue Symbols. In *EUROCRYPT 2013*.

[51] Marcel Keller, Gert Læssøe Mikkelsen, and Andy Rupp. 2012. Efficient Threshold Zero-Knowledge with Applications to User-Centric Protocols. In *ICITS 2012*.

[52] Kim Zetter, WIRED magazine. 2016. How the top 5 PC makers open your laptop to hackers. https://www.wired.com/2016/05/2036876/. (2016).

[53] Junzuo Lai, Robert H. Deng, Changshe Ma, Kouichi Sakurai, and Jian Weng. 2016. CCA-Secure Keyed-Fully Homomorphic Encryption. In *PKC 2016*.

[54] Benoît Libert and Jean-Jacques Quisquater. 2003. Efficient Revocation and Threshold Pairing-based Cryptosystems. In *PODC 2003*.

[55] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *USENIX 2018*. 973–990.

[56] Philip D. MacKenzie and Michael K. Reiter. 2001. Networked Cryptographic Devices Resilient to Capture. In *S&P 2001*.

[57] Philip D. MacKenzie and Michael K. Reiter. 2004. Two-party Generation of DSA Signatures. *Int. J. Inf. Sec.* (2004).

[58] Claudio Marforio, Nikolaos Karapanos, Claudio Soriente, Kari Kostiainen, and Srdjan Capkun. 2013. Secure Enrollment and Practical Migration for Mobile Trusted Execution Environments. In *SPSM'13*.

[59] Brian McGillion, Tanel Dettenborn, Thomas Nyman, and N. Asokan. 2015. Open-TEE - An Open Virtual Trusted Execution Environment. In *TrustCom 2015*.

[60] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. 2013. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *S&P 2013*.

[61] Lasse Øverlier and Paul F. Syverson. 2006. Locating Hidden Servers. In *S&P 2006*.

[62] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT '99*.

[63] Torben P. Pedersen. 1991. A Threshold Cryptosystem without a Trusted Party (Extended Abstract). In *EUROCRYPT '91*.

[64] Torben P. Pedersen. 1991. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO '91*.
[65] Roel Peeters, Svetla Nikova, and Bart Preneel. 2008. Practical RSA threshold decryption for things that think. In *WISSec 2008*.
[66] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The Loopix Anonymity System. In *USENIX 2017*.
[67] Manoj Prabhakaran and Mike Rosulek. 2008. Homomorphic Encryption with CCA Security. In *ICALP 2008*.
[68] Tal Rabin. 1998. A Simplified Approach to Threshold and Proactive RSA. In *CRYPTO '98*.
[69] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell. 2005. Stronger Password Authentication Using Browser Extensions. In *USENIX 2005*.
[70] Ravi S. Sandhu and Xinwen Zhang. 2005. Peer-to-peer access control architecture using trusted computing technology. In *SACMAT 2005*.
[71] Alfredo De Santis, Giovanni Di Crescenzo, and Giuseppe Persiano. 2000. Necessary and Sufficient Assumptions for Non-iterative Zero-Knowledge Proofs of Knowledge for All NP Relations. In *ICALP 2000*.
[72] Victor Shoup. 2000. Practical Threshold Signatures. In *EUROCRYPT 2000*.
[73] Victor Shoup and Rosario Gennaro. 2002. Securing Threshold Cryptosystems against Chosen Ciphertext Attack. *J. Cryptology* (2002).
[74] Patrick P. Tsang, Man Ho Au, Apu Kapadia, and Sean W. Smith. 2010. BLAC: Revoking Repeatedly Misbehaving Anonymous Users without Relying on TTPs. *TISSEC* (2010).
[75] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. 2016. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In *CCS 2016*.
[76] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. In *SIGSAC 2017*.
[77] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *FOCS '86*.

A Tandem Details and Extensions

A.1 The Joye-Libert Encryption Scheme

For completeness, we summarize here the Joye and Libert additively homomorphic encryption scheme [50] that TANDEM uses. These definitions are reproduced from the original paper [50].

Definition 3 ([50])**.** Let q be an odd prime and let $n \geq 2$ such that $n|(q-1)$. Then we define the *n-th power residue symbol modulo q*, $\left(\frac{a}{q}\right)_n$, as the smallest (in terms of absolute number) representation of $a^{\frac{q-1}{n}} \mod q$.

Definition 4 ([50])**.** The Joye-Libert additively homomorphic encryption scheme is given by the following algorithms.

– $\mathsf{HE.Keygen}^+(1^\ell, \beta)$. The key generation algorithm takes as input a security parameter ℓ and an integer $\beta \geq 1$ indicating the bit-size of the message space. Generate primes $r, s \equiv 1 \pmod{2^\beta}$, and set the modulus $n = rs$. Finally, pick $y \in_R \mathbb{J}_n \setminus \mathbb{QR}_n$, i.e., a number whose Jacobi symbol is 1, but that is not a quadratic residue. Return the public key $pk = (n, y, \beta)$ and the private key $sk = r$. The message space will be $\{0, \dots, 2^\beta - 1\}$
– $\mathbf{E}^+_{pk}(m)$. To encrypt a message $m \in \{0, \dots, 2^\beta - 1\}$ against public key $pk = (n, y, \beta)$ pick a random $r \in \mathbb{Z}^*_n$ and return the ciphertext $c = y^m r^{2^\beta} \mod n$.
– $\mathbf{D}^+_{sk}(c)$. Given a ciphertext c and a private key $sk = r$, return the message $m \in \{0, \dots, 2^\beta - 1\}$ such that

$$\left[\left(\frac{y}{r}\right)_{2^\beta}\right]^m = \left(\frac{c}{r}\right)_{2^\beta} \pmod{r}.$$

See Joye and Libert [50] for an efficient algorithm to finding m.

A.2 Using More Than One Tandem Server

So far we described TANDEM for use with a single TANDEM server (TS). In this appendix we show how we can extend TANDEM to support multiple TSs at the same time. Depending on the configuration, using more than one TS provides robustness against TSs that are off-line or deny service to the user, and protects the user against TSs that violate the security assumptions by colluding with an attacker.

The idea is to use a t-out-of-n Shamir secret-sharing of the server's secret x_S and to give each TS one secret-share of x_S. Now, the user needs to cooperate with t TSs to use her key. Note that the user's secret x_U is still needed, so no coalition of TSs can, by themselves, collude to use the user's key. An attacker that corrupts the user's device, and thus learns x_U, needs to additionally collude with t TSs to use the user's key. Therefore, using $t > 1$ strengthens key security if the security assumption for TSs cannot be fully guaranteed. If $t < n$ some of the TSs need not participate when the user wants to use her key, giving the user robustness against denial of service by at most $n - t$ non-cooperating TSs.

The threshold-cryptographic protocols that we have considered in this paper, i.e., Schnorr signatures, ElGamal encryption and credential schemes, can be extended to allow Shamir secret sharing of the TSs keys.

When using multiple TSs, users must decide which ones compute the TCP correctly. The user can directly verify the zero-knowledge proofs in for example Schnorr signatures and the credential protocols (see next section) to check that the TS behaved correctly. The ElGamal encryption scheme can be extended so the TS proves correct behavior.

We now show how to modify the TANDEM protocols to support multiple TSs. During RegisterUser the user registers with each of the n TSs. However, in step 2, the n TSs jointly compute a t-out-of-n Shamir secret-sharing polynomial $f(X) = a_0 + a_1 X + \ldots + a_{t-1} X^{t-1}$ with $a_i \in \mathbb{Z}_p$, for example using Pedersen's verifiable secret-sharing protocol [63]. The server's secret x_S shared by the n TSs then equals a_0. TANDEM server i stores $x_S^i = f(i)$ and sends the corresponding ciphertext $\overline{x}_S^i$ to the user together with the range proof. Each TS uses its own key-pair for the additively homomorphic encryption scheme.

To obtain a one-time-use key-share token the user runs ObtainKeyShareToken with each of the TSs in parallel. To support multiple TSs, the user changes how it chooses value of δ_i that it uses for the ith TS in step 7. The user creates a new, random, secret-sharing polynomial $f'(X) = \delta + b_1 X + \ldots + b_{t-1} X^{t-1}$ for $\delta, b_i \in_R \mathbb{Z}_p$. Then, the user picks $\delta_i \in_R [2^{\ell_\mu}, 2^{\ell_\mu + 1})$ subject to the constraint that $\delta_i = f'(i) \pmod p$. Thereafter, the user continues as before.

To use her key, she picks t tokens, and runs GenShares with each of the corresponding TSs. By choice of the δ_i and x_S^i the TSs operate on the t-out-of-n shared secret $x_S + \delta \pmod p$. The user then uses $x_U - \delta \pmod p$ so that the resulting secret still is $x = x_U + x_S$.

B Attribute-Based Credentials

In this section we specify the TCP protocols used for issuance and showing of BBS+ credentials. Next, we prove that they satisfy the TCP security and privacy conditions required for TANDEM.

B.1 The Full TCP Protocols

In the body of the paper we argued that the following proof of knowledge

$$\mathsf{PK}\{(x, s') : U = B^{s'} B_0^x\}.$$

in the BBS+ issuance protocol can be converted to a threshold-cryptographic version following the ideas of the threshold Schnorr protocol in Fig. 2. We show the full, non-interactive version of this protocol in Fig. 4. The issuer creates a nonce nonce to ensure freshness. In these protocols $H : \{0,1\} \to \mathbb{Z}_p$ is a cryptographic hash function which we will later model as a random oracle.

To enable BBS+ issuance and verification, the TS must participate in two protocols, one for issuance, shown in Fig. 4, and one very similar protocol for showing credentials. We summarize the TS's side of these protocols.

Protocol 6. *The TCP.TS protocol run by the TS for BBS+ schemes is as follows. The TS takes as input the randomized secret share $\tilde{x}_S$. First the user indicates to the TS whether she wants the TS to participate in an issuance protocol or a showing protocol. We assume that the group order p and the description of the relevant groups are known to the TS.*

Issuance. *If the user indicates an issuance protocol, they proceed as follows.*

1. *The user sends to the TS the generator B_0, the context $\mathsf{ctx} = U \parallel B \parallel B_0$, a nonce nonce, and the partial commitment $\tilde{U}'$.*
2. *The TS picks $t_S \in_R \mathbb{Z}_p$, and computes the final commitment $\tilde{U} = \tilde{U}' B_0^{t_S}$. Next, the TS computes the challenge $c = H(\tilde{U} \parallel \mathsf{ctx} \parallel \mathsf{nonce})$ and computes its response $r_S = t_S + c \cdot \tilde{x}_S$. The TS sends $\tilde{U}$ and r_S to the user.*

Showing. *If the user instead indicates a showing protocol, they proceed as follows. Let $E_1 \in \mathbb{G}_1, E_2 \in \mathbb{G}_1, E_3 \in \mathbb{G}_T$ represent the commitments in the zero-knowledge proof corresponding to the three respective conjuncts. And let E_3' be the partial commitment without the TS's contribution.*

1. *The user sends to the TS the generator $\hat{e}(B_0, h)$, the context ctx representing the bases and the full proof statement, a nonce nonce, the commitments E_1, E_2 and the partial commitment $\tilde{E_3}'$.*
2. *The TS picks $t_S \in_R \mathbb{Z}_p$, and computes the final commitment $E_3 = E_3' \hat{e}(B_0, h)^{t_S}$. Next, the TS computes the challenge $c = H(E_1 \parallel E_2 \parallel E_3 \parallel \mathsf{ctx} \parallel \mathsf{nonce})$ and computes its response $r_S = t_S + c \cdot \tilde{x}_S$. The TS sends E_3 and r_S to the user.*

TS		**User**		**Issuer**
$\tilde{x}_S \in \mathbb{Z}_p, B_0$, **ctx**		$\tilde{x}_U \in \mathbb{Z}_p, s', g, B, B_0$, **ctx**		U, **ctx**
$t_S \in_R \mathbb{Z}_p$		$t_U, \hat{s}' \in_R \mathbb{Z}_p$		
$u_S = B_0^{t_S}$		$u_U = B_0^{t_U}$		
$\tilde{U} = \tilde{U}' u_S$	$\xleftarrow{\tilde{U}', \text{nonce}}$	$\tilde{U}' = B^{\hat{s}'} u_U$	$\xleftarrow{\text{nonce}}$	nonce $\in_R \{0,1\}^{128}$
$c = H(\tilde{U} \parallel \text{ctx} \parallel \text{nonce})$				
$r_S = t_S + c \cdot \tilde{x}_S$	$\xrightarrow{\tilde{U}, r_S}$	$c = H(\tilde{U} \parallel \text{ctx} \parallel \text{nonce})$		
		$s_{s'} = \hat{s}' + c \cdot s'$		
		$r_U = t_U + c \cdot \tilde{x}_U$		
		$r = r_U + r_S$	$\xrightarrow{c, s_{s'}, r}$	$\tilde{U}'' = U^{-c} \cdot B^{s_{s'}} \cdot B_0^r$
				$c \overset{?}{=} H(\tilde{U}'' \parallel \text{ctx} \parallel \text{nonce})$

Fig. 4. Full details of the non-interactive proof of knowledge of the user's commitment $U = B^{s'} B_0^{x_U} B_0^{x_S}$ in the BBS+ TCP issuance protocol, where ctx $= U \parallel B \parallel B_0$ captures the statement to be proven. The Tandem server only knows $\tilde{x}_S$ and the user knows $\tilde{x}_U$ and the randomness s' (recall $\tilde{x}_S$ and $\tilde{x}_U$ are the respective outputs of the GenShares protocol). The TS effectively creates a zero-knowledge proof of knowing $\tilde{x}_S$.

B.2 TCP Security and Privacy

We now prove that the TCP protocol in Protocol 6 is TCP secure (see Game 7.2). In this case, the SP models the role of credential issuer and credential verifier. To win in the challenge phase, the adversary should convince the SP (acting as verifier) that it holds a valid credential. Of course, this only makes sense if this credential was issued against a key $x = x_U + x_S$ protected by the TS. Therefore, the SP will only accept in the challenge phase if it can confirm that it issued this credential on a TS protected key x.

To enable tracking of protected credentials, we assume that the credential includes a random attribute $a_1 \in_R \mathbb{Z}_p$ known to the issuer. In most instances of attribute-based credential schemes such an attribute either already exists (e.g., a user identifier) or can be cheaply added. The particular choice of random attribute is not important. What matters is that all protected credentials have a set of attributes that differentiates them from non-protected credentials.

To track which credentials are protected, the SP and TS proceed as follows. For every credential that the SP issues, it checks that the challenge c in the proof of knowledge of the commitment U was computed by the TS when running TCP.TS in issuance mode. As a result, the SP can be sure that the TS-protected keyshare $\tilde{x}_S = x_S + \delta$ is included in U. If this is the case, the SP stores the (random) attribute $a_1 \in_R \mathbb{Z}_p$ for this credential in a list.

During the challenge phase of the TCP security game, the adversary must demonstrate possession of a TS-protected credential. To this end, the adversary will reveal the attribute a_1 as part of the showing protocol. The SP will accept, and the adversary will win, if attribute a_1 matches a protected credential.

Theorem 3. *Provided that the discrete logarithm problem is hard in $\mathbb{G}_1$ and BBS+ credentials are unforgeable, the TCP.TS protocol in Protocol 6 is TCP secure (see Game 7.2) in the random oracle model for H.*

In the proof we show that if the adversary successfully manages to prove possession of a BBS+ credential with a key that is shared with the TS, then it must also be able to break discrete logarithms. To reduce to the DL problem, the challenger will act as if the (unknown) discrete logarithm equals x_S. Since the TS doesn't know x_S, we use the random oracle to simulate the proofs in which the TS is involved.

In the challenge phase, we then extract the secret, and therefore the discrete logarithm from the successful prover. Note that the TS does not participate during the challenge phase, so there are no conflicts with it simulating the proofs that it then also tries to extract. As a result, we break the discrete logarithm.

Proof. Suppose that adversary $\mathcal{A}$ can convince the SP that it possesses a credential containing a valid (i.e., protected by the TS) attribute a_1. Since we assumed that forging BBS+ credentials is hard, $\mathcal{A}$ must have proven possession of one of the credentials issued by the TS. We will build an adversary $\mathcal{B}$ to break the discrete logarithm assumption in $\mathbb{G}_1$.

Let (G, Z) be a discrete logarithm instance in $\mathbb{G}_1$. The goal of $\mathcal{B}$ is to find z such that $Z = G^z$. To this end, $\mathcal{B}$ sets $B_0 = G$ and proceeds as if $x_S = z$. Since $\mathcal{B}$ can no longer complete the zero-knowledge proofs in TCP.TS when it receives TCP(δ) queries, it simulates them using the random oracle. Let $Y = ZB_0^{\delta}$ be the randomized public key corresponding to the TS's randomized secret $\tilde{x}_S = x_S + \delta$ (recall that $\mathcal{B}$ does not know $\tilde{x}_S$).

1. For the issuance mode, $\mathcal{B}$ proceeds as follows in step 2. It picks $c, r_S \in_R \mathbb{Z}_p$, sets $u_S = B_0^{r_S} Y^{-c}$ and computes $\tilde{U} = \tilde{U}' u_S$. Finally, it updates the random oracle so that $H(\tilde{U} \parallel \mathsf{ctx} \parallel \mathsf{nonce})$ equals c.
2. For the showing mode, $\mathcal{B}$ proceeds similarly. In step 2 it picks $c, r_S \in_R \mathbb{Z}_p$, and computes $E_3 = E_3' \hat{e}(B_0^{r_S} Y^{-c}, h)$. Finally, it updates H such that $H(E_1 \parallel E_2 \parallel E_3 \parallel \mathsf{ctx} \parallel \mathsf{nonce})$ equals c.

In both cases, the probability that patching H fails is negligible, because the input to H is random from the perspective of $\mathcal{A}$.

To recover the discrete logarithm of Z we must know the user's key x_U such that the credential contains $x = x_S + x_U$. In fact, nothing limits the user from using a different x_U for each credential. Therefore, we extract this user-related part as follows.

During the issuance protocol with the SP, the user will interact with the TS to create the proof of knowledge $(c, s_{s'}, r)$ as in Fig. 4 and send it to the SP acting as issuer. Recall, $\mathcal{B}$ controls SP. After receiving the proof, $\mathcal{B}$ will rewind $\mathcal{A}$ to the point where it sent $\tilde{U}', \mathsf{nonce}$ to the TS. At this point $\mathcal{A}$'s randomizers $t_U, \hat{s}'$ are fixed. Then $\mathcal{B}$ picks another challenge c' and proceeds as before. $\mathcal{A}$ will send another proof $(c', s'_{s'}, r')$ to the SP. By dividing out the factors the TS created by simulating its proofs, we end up with two traces $(\tilde{U}', c, s_{s'}, r_U)$ and $(\tilde{U}', c', s'_{s'}, r'_U)$ from which $\mathcal{B}$ extracts the user's secret x_U. Therefore, the credential must contain the secret $x = x_U + x_S + \delta$.

Finally, to extract the discrete logarithm of Z, $\mathcal{B}$ rewinds $\mathcal{A}$ in the challenge phase, and extracts the secrets, including the secret key x encoded in the credential. It looks up the corresponding value x_U extracted when issuing this credential and returns the discrete logarithm $z = x - x_U - \delta$ of Z with respect to G. □

In the full TANDEM security game (Game 7.1) we require additionally that the protocol is only computable by U^* so that the challenger can confirm a win. This property is trivially satisfied for any protocol that identifies the user, such as Schnorr's proof of identification or signature schemes. In those cases, the SP simply asks for a new proof of identity or a signature on a new message; and security relates to the non-impersonation property and non-forgeability properties of the underlying schemes.

When applying TANDEM to protocols where the user is anonymous with respect to the SP, e.g., when showing a credential, this condition requires a little bit more work to verify. Formally, we require that the protocol P run by the SP in the challenge phase takes as extra input the identity of a user U^*. The protocol P will only accept if the SP can verify the identity of user U^*.

To facilitate this check for the BBS+ protocols above, we proceed as follows. First, for every issuance that the adversary participates in, it can choose to reveal the token owner's identity U to the SP. When the adversary does, the SP links the random attribute a_1 to user U. Two, during the challenge phase, the adversary reveals the attribute a_1. If the SP recorded this attribute as corresponding to user U^*, the adversary wins.

The TCP privacy property (see Game 7.4) follows by inspection. First, note that the key $\tilde{x}_S$ that users send to the TS (controlled by the adversary) information theoretically hides the user's identity. Next, the values that the users send in step 1 of the TCP protocol (see Protocol 6 above) are independent from any user secret. Therefore if a coalition of SP and TS can distinguish users, this must be because of what the user sends to the SP. The theorem follows.

Theorem 4. *The $\mathsf{TCP.TS}$ protocol in Protocol 6 is TCP Private (see Game 7.2) against honest SPs. Moreover, provided the issuance and showing runs do not identify the user to the SP, then these protocols are also TCP private against colluding SPs.*

B.3 Rate-limiting in ABCs

Anonymous users can use the cover of privacy to misbehave, negatively impacting the system. ABC systems are not exempt from such misbehavior. Suppose, for example, that a user shares her "I am older than 18" credential with many under-aged users who do not hold such a credential. Then, those under-aged users can incorrectly convince service providers that they are over 18 years of age. If this happens often, service providers can no longer rely on these credentials to verify that a user is older than 18.

To limit such misbehavior, ABCs could benefit from rate-limiting. One method to limit abuse is to rate-limit credentials by ensuring that credentials can only be used a limited number of times. For instance, solutions such as n-times anonymous credentials [22] use custom cryptographic techniques to construct a special type of ABC that can be used only n times per epoch.

TANDEM can achieve a similar type of rate-limiting *without* modifying the underlying cryptographic construction of ABCs. To rate-limit use of a system, the TS enforces a per-user and per-epoch limit q on the number of tokens it issues per user and per epoch. As a result, no credential can be shown more than q times per epoch. This approach limits *all* credentials associated to a user's key. If desired, TANDEM can equally be applied on a per-credential basis.

This rate-limiting strategy *requires* that all users use TANDEM. However, recall that the SPs (issuers and verifiers) cannot detect the use of TANDEM, allowing users to forego sharing their keys with the TS, thus avoiding the rate limit. To enable the TS to enforce a rate-limit on all credentials, issuers must only issue credentials on keys that are shared with the TS.

A small change to the threshold-cryptographic version of the issuance protocol enables the issuer to confirm that users use TANDEM. To signal its involvement, the TS signs the challenge c and sends the signature σ to the user. The user forwards the challenge to the issuer. The issuer verifies the signature σ. If the proofs are correct, then the user's key was shared with the TS and the issuer signs the credential.

C Threshold ElGamal Decryption

As a second example, we show how TANDEM can be easily applied to ElGamal decryption. Let $x = x_U + x_S$ be the secret-shared private key and $h = g^x$ the corresponding public key. The ElGamal encryption of a message $m \in \mathbb{G}$ is given by $(c_1, c_2) = (g^r, m \cdot h^r)$ where $r \in_R \mathbb{Z}_p$ is an ephemeral key.

To threshold-decrypt a ciphertext (c_1, c_2) the user and the TS proceed as follows. They first run GenShares, so that the user and the TS hold the respective shares $\tilde{x}_U$ and $\tilde{x}_S$ such that $x = \tilde{x}_U + \tilde{x}_S$. Then the user sends c_1 to the TS which computes $\alpha = c_1^{-\tilde{x}_S}$ and sends it back to the user. The user can now recover the message as $m' = c_2 \alpha c_1^{-\tilde{x}_U}$. Note that the TS never learns the value of the message. In fact, the user could blind c_1 before sending it, to ensure that the TS cannot recognize the ciphertext either.

D Proofs of Lemmas

Proof of Lemma 1. Whenever a ciphertext c_i is selected by the TS for opening, the TS checks that it and the corresponding randomizers κ_i, μ_i, ξ_i, and r_i are as in equation (3) and that $\mu_i < 2^{\ell_\mu}$, and hence as stated in the theorem.

Since the TS checks k tuples, every adversary needs to include at least k correct tuples in its set of $2k$ tuples. If no index i^* exists for the remaining tuples, then all k of them were incorrectly formed. The probability that none of these k bad tuples were selected during the cut-and-choose protocol is $1/\binom{2k}{k}$. □

Proof of Lemma 2. From Lemma 1 we know that with probability $1 - 1/\binom{2k}{k}$ there exists i^* and μ^*, x_S such that

$$\mathbf{D}^+_{sk}(c_{i^*}) = x_S + \mu^*$$

Let $c = \mathbf{E}^+_{pk}(\alpha)$. From equation (4) we know that:

$$c = c_{i^*} \cdot \mathbf{E}^+_{pk}(\gamma_{i^*}; \kappa_{i^*})$$

By decrypting we find that $\alpha = x_S + \mu^* + \gamma_{i^*} \pmod N$. Moreover, $\mu^* < 2^{\ell_\mu}$ (by Lemma 1), $x_S < p < 2^{\ell_\mu}$ (by construction) and $\gamma_{i^*} < 2^{\ell_\mu+1}$ as checked by the TS. Since $\ell_\mu = \lceil \log p \rceil + \ell + \log(k+1) + 2$ and $N > 2^{\ell_\mu+2}$, we have that $\alpha = x_S + \mu^* + \gamma_{i^*}$ as integers, and thus c is a proper randomization, with randomizer $\mu^* + \gamma_{i^*} < 2^{\ell_\mu+2}$, of x_S as well. □

E Constructing Correctness Proof of $\overline{x_S}$

In this section we describe the details of the range proof of $\mathbf{D}^+_{sk}(\overline{x_S})$ in the RegisterUser protocol. The range proof ensures that the TS cannot recognize anonymous users by constructing specially crafted versions of $\overline{x_S}$ as explained earlier. When using a homomorphic encryption scheme that supports zero-knowledge proofs, such as Paillier's encryption scheme, we can use standard techniques, see for example the bitwise technique by Bellare and Goldwasser [13], to prove that $\mathbf{D}^+_{sk}(\overline{x_S})$ is at most 2ℓ bits (which is a sufficient proxy for p in our schemes).

In our implementation, however, we use Joye and Libert's encryption scheme which does not readily admit zero-knowledge proofs. Therefore, we instantiate the range proof using a construction that consists of two parts.

I. The TS constructs a commitment C to x_S using a commitment scheme whose message space is at least as big as the plaintext space of the encryption scheme. The TS then uses a traditional zero-knowledge proof to show that the value x_S committed in C is smaller than p.

II. Next, the TS uses a cut-and-choose technique to show that C commits to $\mathbf{D}^+_{sk}(\overline{x_S}) = x_S$.

The details are as follows. The user and TS take $\overline{x_S}$ as input. The TS takes as private input x_S and the randomizer κ used to construct $\overline{x_S}$. Let $\overline{\mathbb{G}}$ be a cyclic group of order $\overline{p}$ generated by $\overline{g}$ such that $\overline{p} > N$ (recall, N is the size of the plaintext domain of the homomorphic encryption scheme). Let $\overline{h}$ be another generator of $\overline{\mathbb{G}}$ such that the discrete logarithm of $\overline{h}$ with respect to $\overline{g}$ is unknown. We use this group to create a commitment scheme with a large message space.

The full protocol has 7 steps. Part I is represented by step 1, whereas part II is represented by the cut-and-choose technique in steps 2 – 7. If at any step a verification fails, the protocol is aborted. The cut-and-choose technique is very similar to the construction we use in the ObtainKeyShareToken and GenShares protocols. Let k be the difficulty level of the cut-and-choose protocol.

1. The TS picks $r \in_R \mathbb{Z}_{\overline{p}}$, and computes the commitment $C = \overline{g}^{x_S}\overline{h}^r$. Next, the TS creates a non-interactive proof that the commitment C contains key-share x_S of the correct size:
$$PK\{(x_S, r) : C = \overline{g}^{x_S}\overline{h}^r \wedge x_S \in [0, p)\}, \qquad (6)$$
and sends C and this proof to the user. This proof can be implemented using a standard technique like the bitwise commitment technique of Bellare and Goldwasser [13]. The user checks the correctness of the proof.
2. The user randomly chooses a subset $\mathcal{D} \subset \{1, \dots, 2k\}$ of cardinality k. She commits to $\mathcal{D}$ by picking $\theta \in_R \{0,1\}^\ell$ and sending $\Delta = \mathsf{ExtCommit}(\mathcal{D}, \theta)$ to the TS.
3. The TS picks randomizers $\mu_1, \dots, \mu_{2k} \in_R \{0,1\}^{\ell_\mu}$ and $\kappa_1, \dots, \kappa_{2k} \in_R \mathcal{R}$ to construct ciphertexts, and $r_1, \dots, r_{2k} \in \mathbb{Z}_{\overline{p}}$ to create commitments. Then, the TS sets:
$$\begin{aligned} c_i &= \mathbf{E}^+_{pk}(\mu_i; \kappa_i) \\ C_i &= \overline{g}^{\mu_i}\overline{h}^{r_i} \end{aligned} \qquad (7)$$
for $i = 1, \dots, 2k$. Finally, the TS sends the ciphertexts $c_1, \dots, c_{2k}$ and commitments $C_1, \dots, C_{2k}$ to the user. The commitments are computationally binding and information theoretically hiding. (Contrary to the ObtainKeyShareToken protocol, the TS can safely send the ciphertexts, because the user cannot decrypt them.)
4. The user sends the subset $\mathcal{D}$ and the commitment randomizer θ to the TS.
5. If $\Delta = \mathsf{ExtCommit}(\mathcal{D}, \theta)$, then the TS sends $(\mu_i, \kappa_i, r_i)_{i \in \mathcal{D}}$ to the user (otherwise, it aborts). The user verifies that the values c_i, C_i for $i \in \mathcal{D}$ satisfy equation (7). Moreover, the user checks that $\mu_i < 2^{\ell_\mu}$ for $i \in \mathcal{D}$.
6. Next, the TS computes
$$\gamma_i = \mu_i - x_S, \quad \rho_i = r_i - r, \quad \nu_i = \kappa_i \kappa^{-1}$$
for $i \notin \mathcal{D}$, and sends them to the user.
7. Finally, the user checks that
$$\begin{aligned} c_i &= \overline{x_S} \cdot \mathbf{E}^+_{pk}(\gamma_i; \nu_i) \\ C_i &= C \cdot \overline{g}^{\gamma_i}\overline{h}^{\rho_i} \end{aligned} \qquad (8)$$
and that $0 \le \gamma_i < 2^{\ell_\mu}$ for $i \notin \mathcal{D}$, and accepts the proof if all verifications are correct.

Lemma 3. *If the user does not reject in the above protocol, then with probability $1 - 1/\binom{2k}{k}$ we have that $\mathbf{D}^+_{sk}(\overline{x_S}) \in [0, p)$ as required.*

Proof. From the zero-knowledge proof in step 1, we know that the TS knows an opening α', r' of $C = \overline{g}^{\alpha'}\overline{h}^{r'}$ such that $0 \le \alpha' < p$. We complete the proof by showing that $\alpha' = \mathbf{D}^+_{sk}(\overline{x_S})$.

We continue as per Lemma 1 and Lemma 2. We restate them here for completeness. First, along the lines of Lemma 1, with probability $1 - 1/\binom{2k}{k}$ there exists an index i^* such that the TS knows an opening μ^*, r^* such that:
$$\begin{aligned} \mu^* &= \mathbf{D}^+_{sk}(c_{i^*}) < 2^{\ell_\mu} \\ C_{i^*} &= \overline{g}^{\mu^*}\overline{h}^{r^*}. \end{aligned} \qquad (9)$$
The user checks that the TS knows an opening for the k pairs that are opened by the TS in step 4. So, the TS must include at least k pairs for which it knows a correct opening. Suppose, for contradiction, that the index i^* does not exist, i.e., that the remaining k pairs are incorrect or cannot be opened by the TS. Since the protocol completed, the user did not detect foul play. This situation can only occur if the TS correctly guesses the set $\mathcal{D}$ in advance. Since the TS does not learn anything about $\mathcal{D}$ before step 3, the probability that none of the remaining pairs is correct is $1/\binom{2k}{k}$, as required.

Assume now that this index i^* as required above exists. We use this to show that C commits to $\mathbf{D}^+_{sk}(c)$, i.e., that $\alpha' = \mathbf{D}^+_{sk}(c)$. From equation (8) we know that:
$$C_{i^*} = C \cdot \overline{g}^{\gamma_{i^*}}\overline{h}^{\rho_{i^*}}$$

so, by using equation (9) and equating exponents, we find that $\mu^* = \alpha' + \gamma_{i^*} \pmod{\overline{p}}$. We know from the zero-knowledge proof that $\alpha' < p$ and by direct inspection that $\gamma < 2^{\ell_\mu}$ therefore, the equality holds over the integers as well, and we have

$$\mu^* = \alpha' + \gamma_{i^*} < 2^{\ell_\mu + 1} < N. \tag{10}$$

From equation (8) we also know that:

$$c_{i^*} = \overline{x_S} \cdot \mathbf{E}^+_{pk}(\gamma_{i^*}; \nu_{i^*})$$

By decrypting and using equation (9) we find that:

$$\mu^* = \mathbf{D}^+_{sk}(\overline{x_S} \cdot \mathbf{E}^+_{pk}(\gamma_{i^*}; \nu_{i^*})) = \mathbf{D}^+_{sk}(\overline{x_S}) + \gamma_{i^*} \pmod{N}.$$

Substituting μ^* from equation (10) and substracting γ_{i^*} shows that $\alpha' = \mathbf{D}^+_{sk}(\overline{x_S}) \pmod{N}$, and therefore, by size of α' and $\mathbf{D}^+_{sk}(\overline{x_S}) < N$, that $\alpha' = \mathbf{D}^+_{sk}(\overline{x_S})$ as required. □

In the security proof, we replace $\overline{x_S}$ with the encryption of 0, so that the adversary who has corrupted a user learns nothing about x_S (except what is revealed as a result of the threshold-cryptographic protocol). The following lemma states that we can do so, without the adversary detecting this change.

Lemma 4. *TS can simulate the correctness proof given above such that $\overline{x_S} = \mathbf{E}^+_{pk}(0)$, provided that the encryption scheme is CPA secure and the commitment scheme ExtCommit$(\cdot,\cdot)$ is extractable. This simulation does not require any knowledge of how $\overline{x_S}$ was created.*

This proof uses a sequence of games that interpolates between the situation where the RegisterUser protocol is executed normally, and the situation, where $\overline{x_S}$ is an encryption of 0. This game is as in the security game: the adversary can make RegisterUser, ObtainKeyShareToken, GenShares, and BlockShare queries. It's task is to determine if $\overline{x_S}$ is as in the original protocol, or $\overline{x_S} = \mathbf{E}^+_{pk}(0)$. In particular:

- Game 0. In Game 0, $\overline{x_S}$ is constructed as per the protocol.
- Game 1. We proceed as in Game 0, but simulate the cut-and-choose proof in steps 2 – 7 by extracting $\mathcal{D}$.
- Game 2. As in Game 1, but simulate the zero-knowledge proof in step 1 of the protocol.
- Game 3. As in Game 2, but replace the commitment C by a random commitment.
- Game 4. As in Game 3, but replace $\overline{x_S}$ with an encryption of 0.

We show that each pair of consecutive games is indistinguishable to a polynomial-time adversary. Hence, no adversary can distinguish Game 0 from Game 4, thus proving the lemma.

Proof of Lemma 4. We first show how to simulate the cut-and-choose proof in steps 2 – 7. The adverary sends a commitment Δ to the TS in step 1. We use the extractability of ExtCommit$(\cdot,\cdot)$ to recover $\mathcal{D}$ from Δ (for example, using the random oracle model if it is implemented using a hash-function).

We change how TS acts in step 3. Let $\mathcal{D} \subset \{1, \dots, 2k\}$ be the subset of cardinality k extracted from Δ. For all $i \in \mathcal{D}$ the TS sets c_i and C_i as per equation (7). For other elements, i.e., for $i \in \{1, \dots, 2k\} \setminus \mathcal{D}$, the TS generates $\gamma \in_R \{0, \dots, 2^{\ell_\mu} - 1\}, \rho \in_R \mathbb{Z}_{\overline{p}}, \nu \in_R \mathcal{R}$ and sets c_i and C_i as per equation (8).

In step 4, the adversary reveals $\mathcal{D}$ and θ. If $\Delta = \mathsf{ExtCommit}(\mathcal{D}, \theta)$ then with overwhelming probability, we correctly extracted $\mathcal{D}$. If we correctly extracted $\mathcal{D}$, the TS can open the tuples for $i \in \mathcal{D}$ in step 5 and return γ_i, ρ_i, ν_i for the other elements. Both satisfy the adversary's checks in steps 5 and 7.

Game 0 is indistinguishable from Game 1. The simulated proof can go wrong for two reasons. One, we can fail to extract the disclose set $\mathcal{D}$, but this can only happen with negligible probability. Second, the distribution of γ_is for $i \notin \mathcal{D}$ is not completely correct, however, the size of μ ensures that this difference is statistically hidden from the adversary. So, from the point of view of the adversary, Games 0 and 1 are indistinguishable.

In Game 2 we simulate the zero-knowledge proof in step 1. By construction of the simulator of this proof, the adversary cannot detect this change.

As a result of the changes we made in Game 1, the answers of TS do not depend on the opening of C. So, in Game 3 the TS can generate a random commitment $C \in_R \overline{\mathbb{G}}$. Since Pedersen's commitment scheme is information-theoretically hiding, the adversary cannot detect this change.

In Game 4, the TS sends $\overline{x_S} = \mathbf{E}^+_{pk}(0)$ to the user instead of an encryption of the key-share x_S. As a result of the changes we made in Game 1, the TS can still complete the remaining part of the protocol.

We claim that the adversary $\mathcal{A}$ cannot distinguish Games 3 and 4. Suppose to the contrary that $\mathcal{A}$ *can* distinguish Games 3 and 4. We then show that $\mathcal{A}$ can break the CPA security of the homomorphic encryption scheme.

To do so, we build an adversary $\mathcal{B}$ against the CPA security of the encryption scheme. Recall that $\mathcal{B}$ can make a challenge query on two messages m_0 and m_1. In our case, $\mathcal{B}$ picks $m_0 = x_S$ and $m_1 = 0$. Then, its challenger returns a ciphertext $c_* = \mathbf{E}^+_{pk}(pk, m_b)$ for some bit $b \in_R \{0, 1\}$. Adversary $\mathcal{B}$ needs to guess b.

In RegisterUser queries for the challenge user U^*, adversary $\mathcal{B}$ (which acts as a challenger to $\mathcal{A}$) uses $\overline{x_S} = c_*$. Clearly, if $b = 0$, then $\mathcal{B}$ perfectly simulates Game 3. If $b = 1$, it perfectly simulates Game 4. Therefore, if $\mathcal{A}$ can distinguish between Games 3 and 4, it can break the CPA security of the encryption scheme. □

F Security Proof

In the security proof, the challenger controls the TS and the adversary tries to attack a user. The security proof is a sequence of games. In the final game, the challenger simulates the game using only the TCP oracle of the TCP security game, without knowing the corresponding TS' key share x_S. As a result, any adversary that manages to use the blocked key of that user (or uses that key more often than the rate-limit allows in this epoch) must therefore break the security of the underlying threshold-cryptographic protocol.

We use the following sequence of games:

- Game 0: We play the game as described in the TANDEM Security game, see page 336.
- Game 1: We change the definition of GenShares. The challenger simulates the workings of TS but does not decrypt any ciphertext. Instead, the TS uses the extractability of ExtCommit$(\cdot, \cdot)$ and the Δ_is (from the corresponding ObtainKeyShareToken protocol) to compute the plaintext corresponding to c (without decrypting), which it uses as $\tilde{x}_S$. The TS constructs the range proof of $\overline{x_S}$ in the RegisterUser protocol as before.
- Game 2: We guess the challenge user U^* and we change the definition of RegisterUser for this user: we replace $\overline{x_S} = \mathbf{E}^+_{pk}(x_S)$ by $\overline{x_S} = \mathbf{E}^+_{pk}(0)$.
- Game 3: For all non-challenge users we answer GenShares queries as in the previous game. For U^* the TS simulates the TCP following GenShares using the TCP security oracle (without knowing x_S of U^*).

We then prove the following:

- The adversary cannot distinguish Game 0 from Game 1. We prove that as long as one of the pairs (c_i, Δ_i) is as it should be—and Lemma 1 shows that this is the case with high probability—then we correctly recover the plaintext of c and thus the TS extracts the correct $\tilde{x}_S$, and therefore the TCP is correct as well.
- The adversary cannot distinguish Game 1 from Game 2. We no longer decrypt ciphertexts. Hence, we can use the CPA security of the encryption scheme to show that the adversary cannot distinguish Game 1 from Game 2. More formally, we build a distinguisher that interpolates between Games 1 and 2. The distinguisher makes a query for $m_0 = x_S$ and $m_1 = 0$ to its CPA challenger, and uses the answer as $\overline{x_S}$. Lemma 4 shows the adversary cannot detect this change to RegisterUser. If the CPA challenger returned an encryption of x_S then the distinguisher perfectly simulates Game 1, otherwise it simulates Game 2. We can still answer GenShares queries correctly, since we no longer need to decrypt any ciphertexts.
- The adversary cannot distinguish Game 2 from Game 3 because the TCP oracle simulates the same protocol.
- Finally, if we have an adversary that can win Game 3, then it breaks the security of the TCP because by construction the challenger has no unrevoked respectively unused tokens in the challenge phase for the challenge user U^* because the user is blocked respectively rate-limited.

Proof of Theorem 1. This proof follows the sequence of games highlighted above. Let U^* be the challenge user. We guess this user. If the guess turns out to be incorrect, we repeat the reduction with a new guess.

In Game 1 we change how the TS responds to RunTCP queries, in particular, we change GenShares for the challenge user U^*. The TS (controlled by the challenger) no longer decrypts the ciphertext c revealed in a token, but instead directly recovers the plaintext using the Δ_i and γ_i values. The TS then continues as before.

To enable the TS to answer RunTCP queries without decrypting, the TS stores some extra values whenever $\mathcal{A}$ runs the ObtainKeyShareToken protocol. Whenever the TS issues a credential cred, it extracts the attributes $(\epsilon, sk_U, H(c), H(c_1), \ldots, H(c_k))$ (normally, the TS cannot learn these values). The challenger uses the extractability of ExtCommit$(\cdot, \cdot)$ to find inputs $\mu'_{i_1}, \ldots, \mu'_{i_m}$ and $\kappa'_{i_1}, \ldots, \kappa'_{i_m}$ used to create the unopened commitments $\Delta_{i_1}, \ldots, \Delta_{i_m}$. (The adversary might cheat so that not all Δ_is are true commitments.) By Lemma 1, $m \geq 1$, and there exists

i^* such that the extracted inputs are correct, i.e., $\mu'_{i^*} = \mu_{i^*}$ and $\kappa'_{i^*} = \kappa_{i^*}$. The challenger records the tuple $(U, (i_1, H(c_{i_1}), \mu'_{i_1}, \kappa'_{i_1}), \ldots, (i_m, H(c_{i_m}), \mu'_{i_m}, \kappa'_{i_m}))$ for later use.

We now show how to answer RunTCP queries without needing to decrypt the ciphertexts. The TS initially follows the GenShares protocol. At the start of the protocol, $\mathcal{A}$ proves possession of a fresh, unrevoked signature on the values $(\epsilon, H(c), H(c_1), \ldots, H(c_k))$ to the TS (run by the challenger). Moreover, $\mathcal{A}$ provides $\gamma_1, \ldots, \gamma_k$ and $\nu_1, \ldots, \nu_k$. The TS then checks that these values are correct. If not, it aborts. So far, the challenger follows the protocol.

Now, we deviate from the protocol. By the unforgeability of the blind signatures, this signature must have been obtained by running ObtainKeyShareToken. Hence, the challenger can look up the corresponding tuple $(U, (i_1, H(c_{i_1}), \mu'_{i_1}, \kappa'_{i_1}), \ldots, (i_m, H(c_{i_m}), \mu'_{i_m}, \kappa'_{i_m}))$ from tokens it signed by matching on the hashed ciphertexts. Let $\overline{x_S}$ be the encrypted key share for this user. We use the values $\mu'_{i_1}, \ldots, \mu'_{i_m}$ and $\kappa'_{i_1}, \ldots, \kappa'_{i_m}$ to find the plaintext of one of $c_{i_1}, \ldots, c_{i_m}$ and then use this to compute the plaintext of c.

For $i \in i_1, \ldots, i_m$ test if:

$$c_i = \overline{x_S} \cdot \mathbf{E}^+_{pk}(\mu'_i, \kappa'_i)$$

Let $(i^*, \mu'_{i^*}, \kappa'_{i^*})$ be the tuple that satisfies this equation. By Lemma 1 we know that there must exist an index i^* such that:

$$\begin{aligned} c_{i^*} &= \overline{x_S} \cdot \mathbf{E}^+_{pk}(\mu_{i^*}, \kappa_{i^*}), \\ \Delta_{i^*} &= \mathsf{ExtCommit}((\mu_{i^*}, \kappa_{i^*}), \xi^*), \end{aligned}$$

so this procedure does indeed find such a tuple $(i^*, \mu'_{i^*}, \kappa'_{i^*})$. The plaintext of c_{i^*} thus is $x_S + \mu'_{i^*}$. Therefore, the plaintext of c is $x_S + \mu'_{i^*} + \gamma_{i^*}$ because $c = c_{i^*} \cdot \mathbf{E}^+_{pk}(\gamma_{i^*}, \nu_{i^*})$. Therefore $\tilde{x}_S = x_S + \mu'_{i^*} + \gamma_{i^*} \pmod p$.

Now that the challenger has derived $\tilde{x}_S$ it continues with the TCP as normal. This shows how we can answer RunTCP queries without needing to decrypt the ciphertexts.

Games 0 and 1 cannot be distinguished by the adversary. During ObtainKeyShareToken queries, the TS extracts attributes using rewinding, so this is not detected by the adversary. By Lemma 1 the index i^* exists with overwhelming probability, so the responses of the TS are completely identical for the RunTCP queries made by the adversary.

Let x^*_S be the TS' key-share for the challenge user U^*. In Game 2, we do not send $\overline{x_S} = \mathbf{E}^+_{pk}(x^*_S)$ to the adversary when it makes RegisterUser queries for the challenge user U^*. Instead, we send $\overline{x_S} = \mathbf{E}^+_{pk}(0)$. During RunTCP queries, we first extract the plaintext of c as above, and then add x^*_S. The fact that the TS does not need to decrypt c to answer RunTCP queries together with Lemma 4 shows that the adversary cannot detect this change.

In Game 3, we again change how we answer RunTCP queries for the challenge user U^*. In particular, we will answer this query without using the corresponding key-share x^*_S. Instead, we use the challenge oracle for the TCP security in the query phase. We proceed as before, to find the plaintext δ of c when running GenShares. However, now we use the TCP challenge oracle to run the TCP by making a $\mathsf{TCP}(\delta \mod p)$ query. The TANDEM security challenger relays the messages to the adversary $\mathcal{A}$. After the selection phase, we advance the TCP security challenger to the challenge phase. Moreover, the challenge user U^* cannot obtain new tokens (because U^* is either blocked or rate-limited), and all old tokens have been revoked or used, so we no longer need access to the TCP oracle to answer queries. Finally, in the challenge phase, we relay the messages to the TCP challenger. Then, if adversary $\mathcal{A}$ wins Game 3, it breaks the TCP security of the underlying TCP. Since we assumed this cannot happen, the TANDEM scheme is secure as well. The only difference between Game 2 and Game 3 is that we use the TCP oracle to run the TCP. However, since the TCP oracle uses to correct randomized key, this change is indistinguishable to the adversary. □

G Privacy Proof

In our privacy proof, we reduce an attacker against privacy to an attacker on the blind singing property of the blind signature scheme. We use the following version based on the one by Baldimtsi and Lysyanskaya [10].

Game G.1. *The* blind signature game *is between a challenger controlling an honest user U and an adversary controlling the issuer and the verifier.*

Setup *At the start of the game, $\mathcal{A}$ publishes the public key pk_σ of the signer and outputs all other necessary public parameters.*

Challenge *At some point, the adversary outputs two tuples of attributes m_0 and m_1 on which it wants to be challenged. The challenger picks a bit $b \in_R \{0, 1\}$ and* *randomizers r_0, r_1, and computes two commitments*

$$C_0 = \mathit{Commit}(m_0, r_0)$$
$$C_1 = \mathit{Commit}(m_1, r_1)$$

and proceeds as follows. First, it runs the $\mathit{BSA.BlindSign}(pk_\sigma, C_b)$ protocol with private input m_b, r_b for the user to obtain a signature $(\sigma_b, \tilde{C}_b, \tilde{r}_b)$ on m_b. Next, it runs the $\mathit{BSA.BlindSign}(pk_\sigma, C_{1-b})$ protocol with private input m_{1-b}, r_{1-b} for the user to obtain a signature $(\sigma_{1-b}, \tilde{C}_{1-b}, \tilde{r}_{1-b})$ on m_{1-b}. If both protocols are successful, the challenger sends $(\sigma_0, \tilde{C}_0)$ and $(\sigma_1, \tilde{C}_1)$ to the adversary. Otherwise, it sends nothing.

Guess *Finally, the adversary outputs a guess b' of b. The adversary wins if $b' = b$.*

If no adversary can win this game then the signer cannot recognize the signatures it helped produce.

The computationally hiding commitments in the ObtainKeyShareToken protocol ensure that the TS learns nothing about the unrevealed ciphertexts c and witness ciphertexts c_i which it blindly signs. So, when the user runs GenShares and thereby reveals these ciphertexts, they cannot be directly correlated to a corresponding run of ObtainKeyShareToken. Moreover, the plaintext corresponding to the ciphertexts c_i are fully randomized, so that these too do not reveal anything about the user with which the TS is currently interacting.

The privacy proof folows a sequence of games. Throughout we use a guess i_0, i_1 for the challenge tokens. If this guess turns out to be incorrect when the adversary makes it challenge query, we abort and try again. We first use a sequence of games to show that we can remove identifying information from the ObtainKeyShareToken protocol.

- Game 0 is the Tandem privacy game, see page 337.
- In Game 1, we extract the TS key-shares $x_{0,S}$ and $x_{1,S}$ for users U_0 and U_1 from the TS' proof of knowledge in step I of the RegisterUser protocol, see Appendix E.
- In Game 2, we forge the user's zero-knowledge proof of correct construction of C, the commitment to the epoch, the user's private key sk_U, and the randomized ciphertexts, at the end of ObtainKeyShareToken protocol.
- In Game 3, we use the extractability of $\mathsf{ExtCommit}(\cdot, \cdot)$ to forge the user's cut-and-choose proof in the ObtainKeyShareToken protocol, and send random commitments C_i, Δ_i for $i \notin \mathcal{D}$. However, we honestly construct C as per the protocol.
- In Game 4, for user U_i and the challenge token, we set $c = \mathbf{E}^+_{pk}(x_{i,S} + \delta, \kappa)$ and $c_i = \mathbf{E}^+_{pk}(x_{i,S} + \mu_i, \kappa_i)$ for $i \notin \mathcal{D}$ rather than using $\overline{x_S}$. We commit to c_i for $i \notin \mathcal{D}$ as usual. Lemma 3 shows that with high probability we still follow the protocol correctly.
- In Game 5, we omit $x_{i,S}$ altogether in the construction of the unrevealed c_i, that is, we set:

$$c_i = \mathbf{E}^+_{pk}(\mu_i, \kappa_i) \tag{11}$$

  for all $i \notin \mathcal{D}$. Similarly, we set $c = \mathbf{E}^+_{pk}(\delta, \kappa)$, and use these values to construct C. When answering RunTCP queries, user i adds $x_{i,S}$, which we extract during the RegisterUser protocol, to its long-term secret-share x_U to compensate for this change. The size of the randomizers μ_i and δ ensures that the TS cannot detect this change.
- In Game 6, we replace the user's private key sk_U in the commitment C by the value 0. Because of the hiding property of the Pedersen commitment C (and the fact that we simulate the proof of correct generation of C) ensure the adversary cannot detect this change.
- Finally, in Game 7, we simulate the correct opening of the commitment $\tilde{C}$ in the first step of the GenShares protocol without knowing the randomizer $\tilde{r}$. Note that $\tilde{C}$ itself still commits to the same values as in Game 6. Because we simulate the proof, the adversary does not notice this change.

We are now in the situation where the tokens held by user 0 and 1 are exchangeable. We use this to show that no adversary can distinguish situations $b = 0$ and $b = 1$. We use a sequence of games to interpolate between the two situations. We start from Game 7.

- In Game A, the challenger uses $b = 0$ but otherwise proceeds as in Game 7.
- In Game B, the challenger swaps the signatures of the challenge tokens of users U_0 and U_1. By the blind signature game, the adversary cannot detect this change.
- In Game C, the challenger also swaps the users U_0 and U_1 in the challenge phase. As a result, it perfectly simulates $b = 1$ in Game 7. The privacy property of the threshold cryptographic protocol (with colluding respectively honest SP) ensures that the adversary cannot detect this change.

Since these steps are indistinguishable, no adversary can distinguish the situations $b = 0$ and $b = 1$ in Game 7, and by indistinguishability again, neither can any adversary distinguish these two in the original privacy game.

Proof of Theorem 2. Throughout this proof, we use a guess for the challenge tokens i_0 and i_1 of users U_0 and U_1 respectively. If this guess turns out to be wrong in the challenge step, we abort and try again.

In Game 1, the challenger extracts $x_{0,S}$ and $x_{1,S}$ for users U_0 and U_1. In particular, the challenger runs the knowledge extractor on the proof of knowledge of the RegisterUser protocol, see Equation 6, for each of the users. Since the extractor uses rewinding, the adversary does not detect this.

In Game 2, the challenger forges the proof of knowledge of correctness of the commitment C at the end of the ObtainKeyShareToken protocol for the challenge tokens i_0 and i_1 of users U_0, U_1 respectively. By simulatability, the adversary cannot detect this change.

In Game 3, the challenger extracts the subset $\mathcal{D}$ from the commitment Δ as soon as it receives it. For the two challenge tokens, the challenger (acting as the user) now proceeds as follows. It computes C_i, Δ_i for $i \in \mathcal{D}$ as per the protocol. However, for $i \notin \mathcal{D}$ it lets the unrevealed commitments C_i and Δ_i commit to random values. The proof of knowledge that C commits to the same values as C_i is already forged since a previous step. Because the commitment scheme is computationally hiding, the adversary cannot detect this change. Despite the changes we made, the final token that is stored by the user is exactly the same as in the original ObtainKeyShareToken protocol.

In Game 4 we compute the values c and c_i for user U_j and $i \notin \mathcal{D}$ as

$$c = \mathbf{E}^{+}_{pk}(x_{j,S} + \delta; \kappa)$$
$$c_i = \mathbf{E}^{+}_{pk}(x_{j,S} + \mu_i; \kappa_i)$$

(recall, we extracted $x_{j,S}$ in the RegisterUser phase) instead of $c = \overline{x_S} \cdot \mathbf{E}^{+}_{pk}(\delta; \kappa)$ and $c_i = \overline{x_S} \cdot \mathbf{E}^{+}_{pk}(\mu_i; \kappa_i)$. Lemma 3 shows that with overwhelming probability $\mathbf{D}^{+}_{sk}(\overline{x_S})$ equals the value $x_{j,S}$ we extracted in the RegisterUser protocol, so this change does not modify the adversary's view.

In Game 5 the user omits $x_{j,S}$ in the computation of c and c_i, and instead sets:

$$c = \mathbf{E}^{+}_{pk}(\delta; \kappa)$$
$$c_i = \mathbf{E}^{+}_{pk}(\mu_i, \kappa_i)$$

for the challenge tokens. To compensate for the fact that $x_{j,S}$ is no longer included, the users adds $x_{j,S}$ to x_U. As a result, the threshold cryptographic protocol still completes as before.

The size of the domain from which the μ_is and δ are drawn, ensures that the adversary cannot detect this change when the users uses the token. More formally, the user sends c, c_is, γ_is, and ν_is. However, the last two sets are redundant, they can be computed directly based on c and the c_is. As a result, we can focus on $\delta = \mathbf{D}^{+}_{sk}(c)$ and $\mu_i = \mathbf{D}^{+}_{sk}(c_i)$. By the size of the domain of δ and the μ_is and the size of $x_{j,S}$ the tuples $(\delta, \mu_1, \ldots, \mu_k)$ and $(x_{j,S} + \delta, x_{j,S} + \mu_1, \ldots, x_{j,S} + \mu_k)$ are statistically indistinguishable. As a result no adversary can distinguish Games 4 and 5.

In Game 6, the user omits their private key sk_U from the commitment C by setting

$$C = \mathsf{Commit}((0, \epsilon, H(c), H(c_{i_1}), \ldots, H(c_{i_k})), r).$$

The hiding property of the commitment scheme, the fact that we simulate the proof of correctness of C, and the fact that both challenge users are unrevoked, ensures that the adversary cannot detect this change.

In Game 7, we simulate the proof of knowledge in step 2 of the GenShares protocol for the challenge users. Because the simulation is perfect, the adversary does not notice this change. Note that the commitment $\tilde{C}$ still commits to the correct values.

We now show that no adversary can win Game 7. We again use a sequence of games, but now interpolate between Game A, where the challenger uses $b = 0$ in Game 7, and Game C, where the challenger uses $b = 1$ in Game 7. We construct the intermediate Game B, where user U_0 uses the token i_1 of user U_1 and vice versa. Since the challenge tokens in Game 7 (and thus in Games A, B, and C) do not depend on the user, the TCPs complete correctly as in Game 7.

We first show that Games A and B are indistinguishable. Suppose to the contrary that $\mathcal{A}$ can distinguish Games A and B. We show that we can use $\mathcal{A}$ to build an adversary $\mathcal{B}$ that breaks the blindness property of the signature scheme. In the blind signature game, $\mathcal{B}$ gets oracle access to two users that request a blind signature on one message each. Adversary $\mathcal{B}$ acts as the challenger towards $\mathcal{A}$ in Game 7. At the start of the game $\mathcal{B}$ generates two messages, corresponding to key-share tokens, for which users U_0 and U_1 need a blind signature. It creates:

$$m_0 = (0, \epsilon, H(c_1), \ldots, H(c_k))$$
$$m_1 = (0, \epsilon, H(c'_1), \ldots, H(c'_k)),$$

where the values in the tuples are as in Game 7. Adversary $\mathcal{B}$ sends m_0, m_1 to its blind signature challenger.

During the ObtainKeyShareToken protocols for the challenge tokens, $\mathcal{B}$ simulates its users as follows. When user U_0 is running the blind signature protocol to create the challenge token τ_0, $\mathcal{B}$ uses its challenger of the blind signature game to act as the user. When U_1 runs the blind signature protocol to create token τ_1, $\mathcal{B}$ again uses its blind signature game challenger. Finally, the blind signature challenger outputs two signatures σ_0 and σ_1 on messages m_0 and m_1 respectively. Adversary $\mathcal{B}$ uses σ_0 to construct the key-share token for user U_0, and uses σ_1 to construct the key-share token for user U_1. Note that the blind signature challenger does not output the randomizers for the commitments $\tilde{C}_0$ and $\tilde{C}_1$, but this does not matter as we simulate the proof of knowledge that requires them.

If $b = 0$ in the blind-signature game, $\mathcal{B}$s challenge user first blindly signed m_0, so $\mathcal{B}$ perfectly simulates Game A. If $b = 1$ in the blind-signature game, then $\mathcal{B}$ perfectly simulates Game B. Hence, any distinguisher between Games A and B breaks the blindness property of the blind signature scheme.

We now show that if the TCP scheme is private (with a colluding respectively honest SP), no adversary can distinguish between Games B and C. Suppose to the contrary that adversary $\mathcal{A}$ can distinguish Game B from Game C. We show that we can use $\mathcal{A}$ to build an adversary $\mathcal{B}$ that breaks the privacy property of the TCP scheme. Adversary $\mathcal{B}$ simulates users U_0 and U_1 towards $\mathcal{A}$. The RegisterUser and ObtainKeyShareToken protocols do not involve the users' secrets, so $\mathcal{B}$ computes them directly. We now show how to answer RunTCP queries.

Whenever $\mathcal{A}$ makes a $\mathsf{RunTCP}(U_i, j, \mathsf{in}_U)$ query, $\mathcal{B}$ makes a $\mathsf{RunTCP}(i, \mathsf{in}_U)$ query of its challenger. Distinguisher $\mathcal{B}$'s challenger replies with the TS' key-share $\tilde{x}_S$. Let $\tau = (\sigma, \tilde{C}, \tilde{r}, \epsilon, c, \delta, \kappa, (c_l, \kappa_l, \mu_l)_{l=1,\ldots,k})$ be the jth token of user U_i. Normally, this token dictates a TS key-share unequal to $\tilde{x}_S$, but we can use the random oracle and change the token to ensure that the TS will recover $\tilde{x}_S$. To do so, the adversary sets $\delta' = \delta + (\tilde{x}_S - (\delta \bmod p))$ so that $\delta' \bmod p = \tilde{x}_S$, and then computes $c = \mathbf{E}^+_{pk}(\delta'; \kappa)$. (Note that the size of δ' is correct with overwhelming probability). Adversary $\mathcal{B}$ updates the random oracle to ensure that $H(c') = H(c)$, i.e., the new ciphertexts has the same hash value as the original pairs. Next, $\mathcal{B}$ uses token $\tau' = (\sigma, \tilde{C}, \tilde{r}, c', \delta', (c'_l, \kappa'_l, \mu'_l)_{l=1,\ldots,k})$ to run GenShares with the TS.

The changes to the random oracle ensure that this token is valid. Moreover, the changes to the random oracle succeed with high probability since at no point in the games does the TS learn the inputs to these hash-functions. The TS will derive the correct secret share $\tilde{x}_S$ from τ'. So it runs the correct TCP protocol with the requested user which is simulated by $\mathcal{B}$'s challenger.

To answer $\mathcal{A}$'s challenge queries, $\mathcal{B}$ again uses his challenger and proceeds as above to answer the queries. If $b = 0$ in the TCP privacy game, then $\mathcal{B}$'s first run of RunTCP uses user U_0's key, so $\mathcal{B}$ simulates Game B. Otherwise, if $b = 1$, then $\mathcal{B}$ simulates Game C. So, any adversary $\mathcal{A}$ that can distinguish Games B and C breaks the privacy property of the TCP scheme. This completes the privacy proof. □