Cryptocurrencies with Security Policies and Two-Factor Authentication

Florian Breuer
KIT

Vipul Goyal
CMU and NTT

Giulio Malavolta
MPI-SP

***Abstract*—Blockchain-based cryptocurrencies offer an appealing alternative to Fiat currencies, due to their decentralized and borderless nature. However the decentralized settings make the authentication process more challenging: Standard cryptographic methods often rely on the ability of users to reliably store a (large) secret information. What happens if one user's key is lost or stolen? Blockchain systems lack of fallback mechanisms that allow one to recover from such an event, whereas the traditional banking system has developed and deploys quite effective solutions.**

In this work, we develop new cryptographic techniques to integrate security policies (developed in the traditional banking domain) in the blockchain settings. We propose a system where a smart contract is given the custody of the user's funds and has the ability to invoke a two-factor authentication (2FA) procedure in case of an exceptional event (e.g., a particularly large transaction or a key recovery request). To enable this, the owner of the account secret-shares the answers of some security questions among a committee of users. When the 2FA mechanism is triggered, the committee members can provide the smart contract with enough information to check whether an attempt was successful, and nothing more.

We then design a protocol that securely and efficiently implements such a functionality: The protocol is round-optimal, is robust to the corruption of a subset of committee members, supports low-entropy secrets, and is concretely efficient. As a stepping stone towards the design of this protocol, we introduce a new threshold homomorphic encryption scheme for linear predicates from bilinear maps, which might be of independent interest.

To substantiate the practicality of our approach, we implement the above protocol as a smart contract in Ethereum and show that it can be used today as an additional safeguard for suspicious transactions, at minimal added cost. We also implement a second scheme where the smart contract additionally requests a signature from a physical hardware token, whose verification key is registered upfront by the owner of the funds. We show how to integrate the widely used universal two-factor authentication (U2F) tokens in blockchain environments, thus enabling the deployment of our system with available hardware.

1. Introduction

Bitcoin and blockchain-based cryptocurrencies brought us at the brink of a technological revolution: These systems allow us to bypass the need for centralized trusted entities to run currencies on a large-scale. While their decentralized and borderless nature make them appealing substitutes for Fiat currencies, a burning problem with this approach is the lack of a recovery mechanism if something goes wrong. What happens if one user's key is lost or stolen? There is no bank to call and no authority to rely upon to get your funds back. Indeed, the number of such high profile attacks on cryptocurrencies has been rising steadily. Famous incidents like the Mt. Gox hack [14], the coincheck hack [2], and the Parity Technology code deletion [3] saw funds upwards of several hundred million dollars being lost or stolen. The well-known DAO hack [32] forced Ethereum to adopt a hard fork which created two version of the currency: Ethereum and Ethereum classic. To deal with such problems in future, Ethereum developers have proposed EIP 867. An ethereum improvement protocol (EIP) is the process by which code changes get accepted onto the Ethereum platform. However EIP 867 has proven to be controversial since it will again bifurcate the currency into two versions, the very problem it was trying to solve.

The problems of attackers stealing money or losing credentials are of course not unique to blockchain-based cryptocurrencies. The (traditional) banking industry has been dealing with such issues for decades where a number of mitigating approaches have worked pretty effectively. Examples of common security policies to deal with such attacks include: A waiting period (say 24 hours) for transferring money to a new recipient and an overall daily outgoing transfer limit. Bypassing these restrictions could either require additional verifications (such as two-factor authentication), or may not be allowed at all. Different banks could follow different security policies which might also vary depending on the type of account (business vs personal) and by the state/region and the local laws. This motivates the following question:

> *Can we take the lessons learnt in the traditional banking domain, and apply them fruitfully to blockchain-based systems, without compromising their decentralized nature?*

1.1. System Architecture

We propose a system where each user has the opportunity to delegate the custody of its funds to a smart contract. The security policy (which can be specified by the user itself) is hardwired in the smart contract and it governs its decision. All funds transfer will go through the approval of such a smart contract, which has the power to accept or reject (or even set on hold) each transaction, depending on the specified policy. In case of an exceptional event, such as key theft, or as an additional security safeguard in case of particularly large transactions or transactions to new addresses, the smart contract will require additional

2021 IEEE European Symposium on Security and Privacy (EuroS&P) | 978-1-6654-1491-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/EUROSP51992.2021.00020

DOI 10.1109/EuroSP51992.2021.00020

verification via two-factor authentication (2FA). We stress that one does not need to authenticate all transactions but only a carefully chosen set, at the discretion of the policy specified by the owner of the funds.

In this work we implement the above idea by developing solutions for 2FA mechanisms that can coexist with the decentralized nature of blockchain-based currencies. To be aligned with the philosophy of decentralized system, our solutions are guided by the following design principles.

Distributed Trust. The 2FA must not rely on the existence of a trusted authority. Instead we propose to leverage a hardware token or distribute the trust among a (reasonably-sized) set of parties, which are asked to intervene in case of exceptional event. The members of such a committee can be chosen by the users or can be selected through a consensus protocol (e.g., the miners themselves can play this role). Security must be guaranteed even if a subset of the committee members behaves maliciously.

Reliability and Guaranteed Output Delivery. We require the 2FA mechanism to be resilient against (benign) disconnections and (malicious) denial of service attacks. That is, even if part of the members of the committee go offline or become corrupted, one should still be able to complete the authentication procedure (given that the set of online parties is large enough).

Low Latency. In distributed environments communication is typically expensive, as messages have to be broadcasted to all users in the network. For this reason it is of central importance to minimize the rounds of communication of a 2FA mechanism.

Computational Efficiency. General purpose cryptographic solutions are very powerful but are often computationally intensive. We aim to build a solution based on well-established cryptographic components which is efficient enough to be integrated in real-life systems.

1.2. Two-Factor Authentication Mechanisms

A popular approach to 2FA in the traditional banking system is that of security questions: When the 2FA is triggered, the user is prompted to answer a personal question and the verification process succeeds if the user's answer is correct. If we were to import this idea to the decentralized setting, the first question that arises is where to store the correct answers in absence of a trusted authority. Clearly the smart contract cannot hardwire the correct answers as otherwise they would be public knowledge (smart contracts do not offer any form of privacy). Since answers typically come from a low-entropy distribution, storing the hash of them is also not a good idea since it is vulnerable to off-line dictionary attacks, where an attacker recovers the hash and tests locally his guesses until he succeeds.

Our idea to bypass this obstacle is to *secret share* the correct answers among a set of n parties, in such a way that stealing the secret would require to corrupt at least t members of this committee (for some threshold parameter $t \leq n$). When the 2FA mechanism is triggered, the user is asked to authenticate its transaction τ by providing the correct answers to a set of predefined questions. The user then broadcasts a message to all committee members, who perform some computation locally and output a partial response tied to the transaction τ. The smart contract then collects sufficiently many responses and publicly checks whether the authentication was successful. If this is the case the smart contract allows τ to go through, otherwise it rejects it. In terms of security, we require that the above process does not reveal anything beyond whether the user's guess was correct or not, even if the attacker corrupts a certain subset of committee members. Henceforth, we refer to this procedure as a distributed zero-tester (DZT).[1]

We also consider an alternative setup where we leverage a physical two-factor authentication token. As an example, universal two-factor authentication (U2F) tokens allow a user to sign any message by querying the token. With this tool at hand, authentication is done as follows: The smart contract hardwires the verification key of the U2F token and, when the 2FA mechanism is invoked, sends the user a nonce. The authentication is successful if the user provides a correct signature on the nonce.

1.3. Security Policies

Our system is completely flexible in the policy that is enforced by the smart contract, which can be specified by the owner of the address. A more conservative usage of our system is just as an additional safeguard mechanism: Suspicious transactions (e.g., unusually large amounts or transactions to a new address) can trigger the 2FA mechanism and force the user to provide (human-memorable) answers to some security questions or a signature from a physical token. This gives an additional line of defense even against the catastrophic event where an attacker steals a user signing key.

On the other side of the spectrum, one user may want to set the policy such that some transactions are allowed given *only* the 2FA, i.e., they are not required to be digitally signed. This can be useful in scenarios where a user has lost the signing key for an address and wants to recover the funds: Answering some security questions allows him to transfer the coins to a fresh address (with a newly sampled signing key). However this liberal usage of our system requires careful thoughts. While on the one hand it improves the usability of the currency, on the other hand it opens the doors to a new attack vector: instead of stealing a secret key, an attacker can compromise the address by guessing the answers to some security questions (or stealing a physical token), which is typically a much easier task. This risk can be mitigated by rate-limiting the amount of attempts for *unsigned* transactions to, e.g., once per week. Since each attempt requires the user to post a message over the blockchain, the query limit can be enforced by the smart contract itself, without the need for the committee members to coordinate or to update the code of the physical token.

In this work we mainly focus on the former case, where the 2FA mechanism is invoked *in addition* to the standard digital signature check. An in-depth cost analysis of the security and usability tradeoff of unsigned transactions is beyond the scope of this work.

1. The name comes from the fact that the committee members can jointly check whether an attempt $\tilde{\alpha}$ corresponds to the correct answer α (i.e., $\tilde{\alpha} - \alpha = 0$) and nothing beyond that.

1.4. Our Contribution

In this work we propose a new method to safeguard transactions on blockchain-based cryptocurrencies, inspired by the solutions developed in the (traditional) banking domain. We then develop 2FA mechanisms amenable to the decentralized nature of cryptocurrencies, offering different trade-offs in terms of trust assumptions, physical capabilities, and computational efficiency. Our technical contributions can be summarized as follows.

(1) Definitions. We develop the notion of distributed zero-tester (DZT), the central cryptographic building block that enables efficient 2FA in decentralized system. We give formal definitions (Section 5) for this primitive and we characterize the security requirements with a game-based definition.

(2) Cryptographic Protocol. We propose a cryptographic instantiation of DZT from standard assumptions on bilinear groups (Section 6). The scheme is *round optimal*, has *guaranteed output delivery*, and is *concretely efficient*. Along the way, we develop a new threshold homomorphic encryption scheme for linear predicates from bilinear groups (Section A), which might be of independent interest.

(3) Implementation. We implement the DZT scheme as a smart contract in Ethereum (Section 7). Our performance evaluation shows that, for reasonably-size committee, the resulting 2FA mechanism can be deployed by today's users at minimal additional cost (around 1$ per authenticated transaction). We also implement a U2F-based 2FA mechanism as a smart contract in Ethereum (Section 8), thus enabling 2FA with a hardware token already available to the public. In terms of added cost, our scheme introduces (approximately) an additional 3¢ to 28¢ per transaction, depending on the choice of the curve.

2. Technical Overview

In the following section we give an informal exposition of the techniques developed in this work. We focus on the main goal of designing a crytpographic solution for a DZT: In a DZT a client secret shares an answer α (the primitive naturally extends to the settings of multiple answers) among a set of n parties. When the 2FA is invoked, the client (which has an attempt $\tilde{\alpha}$ in his head) crafts a single query q and sends q (together with the transaction identifier τ) to all parties, who locally compute a response p_i. Given a large enough set of responses $\{p_i\}_{i\in S}$, for some set S of size $|S| = t$, anyone can publicly determine the outcome of the authentication process. The transaction τ is successfully authenticated if and only if $\alpha = \tilde{\alpha}$ and the protocol should not leak any information beyond whether the authentication process succeeded or not.

2.1. A Generic Solution

We first discuss a high-level idea of our solution and then we present an efficient cryptographic instantiation. Threshold fully-homomorphic encryption [22], [11] allows us to compute any function over encrypted data and offers a general solution to our problem: The client can simply compute $\mathsf{Enc}(\mathsf{pk}, \alpha)$ and distribute the shares of the secret key $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ to the committee members. To authenticate a transaction, a user computes $\mathsf{Enc}(\mathsf{pk}, \tilde{\alpha})$ and broadcasts it to all parties. Then the committee members locally compute, using the homomorphic properties of the scheme,

$$c = \mathsf{Enc}\left(\mathsf{pk}, \alpha \stackrel{?}{=} \tilde{\alpha}\right)$$

and compute and output the partial decryption using their share sk_i. The plaintext bit $\{0, 1\}$ of c can be publicly reconstructed using a large enough set of decryption shares and the output of the authentication is set to be this bit.

While this solution satisfies all of our security requirements, it introduces a prohibitively high computational overhead. As of today, implementing a generic fully-homomorphic encryption in an Ethereum smart contract is far off the reach of the current infrastructure. Thus, developing a solution that can be used in today's systems requires us to devise a different strategy.

2.2. A Flawed Attempt

In order to understand our solution, it is instructive to iterate through a simple construction and analyze its pitfalls. In the setup phase, the client samples an ElGamal [18] key pair $(x, h = g^x)$ and encrypts the answer α under such a key, making the resulting ciphertext

$$(c_0, c_1) = (g^r, h^r \cdot g^\alpha)$$

publicly available to all parties. The secret key x is then secret shared (using Shamir scheme [36]) among n parties in such a way that any t shares are sufficient to reconstruct the secret. Let (x_i, i) be the i-th share. To authentication a transaction τ, a user can compute the encryption of its attempt

$$(d_0, d_1) = (g^s, h^s \cdot g^{-\tilde{\alpha}})$$

and broadcast it to all members of the committee. Each party then computes and broadcasts $p_i = (c_0 \cdot d_0)^{x_i}$. Given a set S of responses, the outcome of the authentication can be publicly recovered by checking

$$\prod_{i\in S} p_i^{\lambda_i} \stackrel{?}{=} c_1 \cdot d_1$$

where λ_i is the i-th Lagrange coefficient. Note that if $\alpha = \tilde{\alpha}$, then $c_1 \cdot d_1 = h^{r+s}$ and indeed the above equation holds true since the secret x is reconstructed in the exponent. Unfortunately this solution has multiple flaws. First of all, there is no mechanism that ties the transaction τ to the authentication process so one can authenticate without the knowledge of α by simply replaying a valid ciphertext (c_0, c_1) and swapping the corresponding transaction τ with a new $\tilde{\tau}$. Furthermore, even if a single member provides a malformed answer, the entire mechanism for verification fails.

While these issues can be resolved using non-interactive zero-knowledge proofs (NIZK) [7], there is a more serious problem: If the authentication is not successful, then the responses of the committee members reveals the exact difference $\alpha - \tilde{\alpha}$. This is because

$$\frac{c_1 \cdot d_1}{\prod_{i\in S} p_i^{\lambda_i}} = \frac{h^{r+s} \cdot g^{\alpha-\tilde{\alpha}}}{g^{(r+s)\sum_{i\in S} x_i\lambda_i}} = \frac{h^{r+s} \cdot g^{\alpha-\tilde{\alpha}}}{h^{r+s}} = g^{\alpha-\tilde{\alpha}}$$

which means that the attacker can always guess the correct α with at most two queries. This is a notorious issue in

threshold cryptography, and current approaches to resolve this problem (see, e.g., [20], [17], [30]) require one to add two rounds of interaction. This is not acceptable for us, since we consider settings where communication is expensive (i.e., each message is posted on a blockchain) and thus we aim at a completely *non-interactive* solution.

2.3. Bilinear Maps at Rescue

The above vulnerability comes from the fact that the threshold version of ElGamal encryption does not satisfy the notion of simulation security for equality predicates, i.e., instead of revealing whether two strings are equal or not it reveals the *exact difference*. This problem can be bypassed by resorting to more powerful cryptographic machinery.

Our first observation is that the class of predicates that we need to evaluate is very restricted, so there is hope to build a solution without the full power of fully-homomorphic encryption. Our second observation is that the protocol is already secure if the authentication is successful, so we only need to worry about the case where the answer is not correct, i.e., $\alpha \neq \tilde{\alpha}$.

Our central idea is to revisit the ElGamal-based approach by adding a re-randomization factor to the plaintext of the evaluated ciphertext, which cancels out only if $\alpha = \tilde{\alpha}$. This can be done with the help of bilinear groups. In a bit more detail, in the setup phase, the client publishes

$$(c_0, c_1, c_2, c_3) = (g^r, h^r \cdot g^{\rho\alpha}, g^{\tilde{r}}, h^{\tilde{r}} \cdot g^{\rho}) \in \mathbb{G}_1^4$$

where $h = g^x$. The secret key x is secret shared as before. Note that (c_0, c_1) serves the same role as before, except that there is an additional factor ρ multiplied by the answer α. The role of (c_2, c_3) will be clear in a moment. The authentication begins with the user computing and broadcasting the encryption

$$(d_0, d_1) = (g^s, h^s \cdot g^{-\rho\tilde{\alpha}}) \in \mathbb{G}_1^2.$$

Note that the user does not know the factor ρ but can still compute a valid ciphertext for $-\rho\tilde{\alpha}$ by obliviously raising (c_2, c_3) to the power of $-\tilde{\alpha}$ and then re-randomizing the resulting ciphertext.

Each committee member computes $(c_0 \cdot d_0)^{x_i}$ as before, except that it additionally samples a fresh element $k = g^{\kappa} \in \mathbb{G}_2$ using a random oracle and returns

$$p_i = e\left((c_0 \cdot d_0)^{x_i}, k\right) = e(g, g)^{(r+s)x_i\kappa} \in \mathbb{G}_T$$

as the decryption share. The outcome of the authentication process can be recovered by checking

$$\prod_{i \in S} p_i^{\lambda_i} \stackrel{?}{=} e(c_1 \cdot d_1, k).$$

To see why security is preserved even in the case of a failed authentication, observe that

$$\frac{e(c_1 \cdot d_1, k)}{\prod_{i \in S} p_i^{\lambda_i}} = \frac{h^{(r+s)\kappa} \cdot g^{\kappa\rho(\alpha-\tilde{\alpha})}}{g^{(r+s)\kappa \sum_{i \in S} x_i\lambda_i}} = \frac{h^{(r+s)\kappa} \cdot g^{\kappa\rho(\alpha-\tilde{\alpha})}}{h^{(r+s)\kappa}}$$

so in case $\alpha = \tilde{\alpha}$ the randomization factor $\kappa\rho$ cancels out and the above equality is verified. For the case case $\alpha \neq \tilde{\alpha}$, the factor $\kappa\rho$ completely masks the difference $\alpha - \tilde{\alpha}$ and prevents the attack as described above. Since the value of $k = g^{\kappa}$ is freshly sampled upon each attempt, the randomization factor is pseudorandom for each authentication query. Also note that k can be sampled locally by each party, without the need of any additional interaction.

2.4. Additional Challenges

The above presentation glosses over many important aspects that need to be taken into account when building an efficient protocol. As an example, the above protocol must be augmented with NIZK proofs to make sure that a malicious player cannot deviate from the specification of the protocol. However, using generic NIZK schemes for NP would vanish our efforts to build a practical system. Fortunately we show that our scheme can be slightly tweaked to allow us to implement the required NIZKs using exclusively Schnorr proofs for discrete logarithm equality [35], which results in a concretely efficient scheme.

Another set of challenges arises when implementing the protocol as a smart contract in Ethereum. In order to obtain an efficient protocol, we would like to leverage precompiled instructions to perform bilinear group operations. However the semantic of operations supported by Solidity (the language of Ethereum smart contracts) is very limited: It only supports group operations in the source $\mathbb{G}_1$ and pairing-product equation checks. This turns out to be insufficient to implement the scheme as outlined above. To circumvent this issue, we further modify our construction to make it fully compatible with precompiled instructions in Solidity. This process is not hassle-free: The new scheme requires us to introduce a new (static) assumption over bilinear groups, the *dual* Diffie-Hellman assumption. We then show that such an assumption holds true in the generic group model. We refer the reader to the Section 6.2 sections for further details.

Also the integration of a U2F-based solution in Ethereum introduces some additional complications. As an example, a smart contract cannot implement a randomized algorithm but we need to sample a random challenge to complete the U2F authentication protocol. Instead of a truly random string, we use the hash of a unique identifier of the transaction (to prevent replay attacks) to implement the challenge sampling procedure. We elaborate in more detail in Section 8.

3. Related Work

In the following we survey some related work from the literature.

Multi-Sig Addresses. One of the most prominent approaches to safeguard accounts in cryptocurrencies is the creation of *multi-sig* addresses: A secret key of a digital signature scheme is secret shared among multiple parties using a threshold scheme [24], [29], [20], [16]. Approving transaction requires gathering a large enough set of users to jointly sign it. This means that compromising a single device is no longer sufficient to steal coins.

Our approach complements this idea and adds two important features: (1) Multi-sig address require to gather multiple parties to sign *every* transaction regardless of how small or big it is, whereas in our system a smart security policy can decide when additional verification is

required. *To the best of our knowledge, there has been no previous work which has this feature*. (2) Our system supports also low-entropy secrets (e.g., human-memorable passwords or answers to security questions) that can be used as an additional verification, in case one of the parties loses his credentials.

Password-Protected Wallets. Another approach to improve the usability of blockchain systems is the so-called password-protected wallet [21], where the secret keys are encrypted using a human-memorable password. Unfortunately these systems are vulnerable to exhaustive-search attacks where one can enumerate all plausible passwords and attempt to recover the secret. A DZT can be used to overcome this limitation and to construct a form of *distributed* password-protected wallet across some committee members.

Vaults. Cryptocurrency vaults [33] aim at disincentivizing key theft by delaying the acceptance transactions: Once an illegitimate transaction is placed on the blockchain, the legitimate owner has enough time to prevent the spending using an appropriate *recovery key*. The aim of this mechanism is to reduce the incentive for an attacker to steal keys, as it will likely result in no financial gain. However, it does not address the question of recovering access to an account, in case of unintentional credential losses.

Password-Authenticated Key Exchange (PAKE). An area which is closely related to our DZT is that of threshold PAKE [31], [15], where a human-memorable password is used to authentication a user against a set of servers, who collectively know the password but can be individually corrupted without compromising security. The main difference with respect to our settings is that we do not require to exchange any key, instead we want to tie the authentication process with a transaction chosen by the client. Additionally, we require that the outcome of the authentication process is *publicly verifiable* even by external parties, which is typically not the case for PAKE protocols.

Zero-Testing of ElGamal Ciphertexts. One of our main technical innovation is a new protocol where parties can jointly test whether an ElGamal ciphertext is an encryption of 0, without revealing any additional information. While this is a well-known problem in threshold cryptography [13], known efficient solutions (see, e.g., [20], [17], [30]) are based on a commit-and-prove approach and require at least three rounds of interaction. In contrast, our solution is *completely non-interactive*: Each party broadcasts a single message and the verification procedure is public. This is particularly suitable for settings where communication is expensive, e.g., where parties exchange messages by posting them on a blockchain.

Password-Protected Secret-Sharing (PPSS). A PPSS [4] allows a user to share a secret that can be reconstructed with the sole knowledge of a password. Recent efficient protocols [26] can be used as a substitute for DZT in our 2FA mechanism. However, PPSS would require at least one more round of interaction between the committee members and the client. In contrast, the shares in the DZT can be verified non-interactively and without the need to set up an off-chain communication infrastructure.

Distributed Point Functions (DPF). A DPF [23] (and its generalization to function secret sharing [12]) allows a set of parties to secret-share a point function $f \rightarrow (f_1, \ldots, f_n)$ whose output can be recovered by the output of the local evaluation of each parties, i.e., $f(x) = f_1(x) \oplus \cdots \oplus f_n(x)$. A DZT can be thought as a threshold version of a DPF with a few important differences: (i) In a DZT the output shares of the parties are publicly verifiable, which is in general not the case for DPFs. Furthermore, (ii) in a DZT the output reconstruction is not restricted to be a linear function. Finally, (iii) DZT supports the evaluation of encrypted queries, whereas in a DPF the inputs are typically public.

4. Preliminaries

We denote by $\lambda \in \mathbb{N}$ the security parameter. We say that a function $\mathsf{negl}(\cdot)$ is negligible if it vanishes faster than any polynomial. Given a set S, we denote by $s \leftarrow\!\!\$\, S$ the uniform sampling from S. We say that an algorithm is PPT if it can be implemented by a probabilistic machine running in time polynomial in the security parameter.

4.1. Bilinear Groups

Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be an asymmetric bilinear group of prime order p with an efficiently computable pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ and let $\mathcal{G}$ be the generator of such a group. We denote by (g_1, g_2) the generators of the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and by $g_T = e(g_1, g_2)$ the generator of $\mathbb{G}_T$. In the following we recall the eXternal Diffie-Hellman (XDH) [9] and the Decisional Bilinear Diffie-Hellman (DBDH) [27], [10] problems over bilinear groups.

Assumption 1 (XDH). *Let $\mathcal{G}$ be a bilinear group generator. $\mathcal{G}$ is XDH-hard if for all PPT distinguishers it holds that the following distributions are computationally indistinguishable*

$$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, g_1^x, g_1^y, g_1^{xy}) \approx (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, g_1^x, g_1^y, g_1^z)$$

where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2) \leftarrow\!\!\$\, \mathcal{G}(1^\lambda)$ *and* $(x, y, z) \leftarrow\!\!\$\, \mathbb{Z}_p^*$.

Assumption 2 (DBDH). *Let $\mathcal{G}$ be a bilinear group generator. $\mathcal{G}$ is DBDH-hard if for all PPT distinguishers it holds that the following distributions are computationally indistinguishable*

$$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, g_2^x, g_T^y, g_T^{xy}) \approx (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, g_2^x, g_T^y, g_T^z)$$

where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2) \leftarrow\!\!\$\, \mathcal{G}(1^\lambda)$ *and* $(x, y, z) \leftarrow\!\!\$\, \mathbb{Z}_p^*$.

4.2. Non-Interactive Zero-Knowledge

A non-interactive zero-knowledge (NIZK) proof [7] allows a prover to convince a verifier about the validity of a certain statement without revealing anything beyond that. We recall the syntax in the following.

Definition 1 (NIZK). *Let $\mathcal{L}$ be an NP-language with relation $\mathcal{R}$. A NIZK system for $\mathcal{R}$ consists of the following efficient algorithms.*

$\underline{\mathsf{Setup}(1^\lambda)}$: *On input the security parameter* 1^λ*, the setup algorithm returns a common reference string* crs.

$\underline{\mathsf{Prove}(\mathsf{crs}, \mathsf{stmt}, \mathsf{wit})}$: *On input the common reference string* crs*, a statement* stmt*, and a witness* wit*, the prover algorithm returns a proof* π.

$\underline{\mathsf{Verify}(\mathsf{crs}, \mathsf{stmt}, \pi)}$: *On input the common reference string* crs*, a statement* stmt*, and a proof* π*, the verifier algorithm returns a bit* $\{0, 1\}$.

Correctness requires that for all $\lambda \in \mathbb{N}$ and all pairs $(\mathsf{stmt}, \mathsf{wit}) \in \mathcal{R}$ it holds that

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, \mathsf{stmt}, \mathsf{Prove}(\mathsf{crs}, \mathsf{stmt}, \mathsf{wit}))\right] = 1$$

where $\mathsf{crs} \leftarrow_{\$} \mathsf{Setup}(1^\lambda)$. We recall the definition of zero-knowledge in the following.

Definition 2 (Zero-Knowledge)**.** *A NIZK system for* $\mathcal{R}$ *is zero-knowledge if there exists a PPT algorithm* $(\mathsf{Sim}_0, \mathsf{Sim}_1)$ *such that for all pairs* $(\mathsf{stmt}, \mathsf{wit}) \in \mathcal{R}$ *and for all PPT distinguishers the following distributions are computationally indistinguishable*

$$(\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda), \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{stmt}, \mathsf{wit})) \approx (\mathsf{crs}^*, \pi \leftarrow \mathsf{Sim}_1(\mathsf{crs}, \mathsf{stmt}, \mathsf{td}))$$

where $(\mathsf{crs}^*, \mathsf{td}) \leftarrow \mathsf{Sim}_0(1^\lambda)$.

We require that the protocol satisfies the strong notion of simulation extractability.

Definition 3 (Simulation Extractability)**.** *A NIZK system for* $\mathcal{R}$ *is simulation-extractable if there exists a PPT algorithm* Ext *and a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and all PPT algorithms* $\mathcal{A}$ *it holds that*

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{crs}, \mathsf{stmt}, \pi) & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Sim}_0(1^\lambda) \\ \wedge\ \mathsf{stmt} \notin Q & \pi \leftarrow \mathcal{A}(\mathsf{crs})^{\mathcal{O}(\cdot)} \\ \wedge\ (\mathsf{stmt}, \mathsf{wit}) \notin \mathcal{R} & \mathsf{wit} \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathsf{crs}, \mathsf{td}, \mathsf{stmt}, \pi) \end{array}\right] = \mathsf{negl}(\lambda)$$

where $\mathcal{O}$ *takes as input a (possibly false) statement* stmt *and returns* $\mathsf{Sim}_1(\mathsf{crs}, \mathsf{stmt}, \mathsf{td})$ *and we denote by* Q *the list of queries issued by* $\mathcal{A}$.

4.3. Secret Sharing

We recall the threshold secret sharing scheme from Shamir [36] over a field $\mathbb{F}$ where a randomized algorithm Share takes as input a field element s, a threshold t, and a number of participants n and returns the evaluations of a random $(t-1)$-degree polynomial at the respective points $(1, s_1), \ldots, (n, s_n)$. Then any subset $S \subseteq \{1, \ldots, n\}$ such that $|S| = t$ can recover the secret s by computing

$$s = \sum_{i \in S} \lambda_i \cdot s_i$$

over $\mathbb{F}$, where λ_i is the i-th Lagrange coefficient.

5. Definitions

In the following we present the notion of a distributed zero-tester (DZT) and we provide explicit security and privacy properties. Before delving into the formal definitions, we discuss on a high level our design goals.

5.1. Overview

A DZT allows one to share a secret among a selected committee of users in such a way that later on one can authenticate using the knowledge of such a secret, assuming that a large enough portion of committee members is online. Our definition for a DZT is tailored to distributed environments where communication is expensive and an arbitrary subset of parties may decide not to comply with the protocol specifications. Our definitions (and consequently the instantiation that we propose) are guided by the following design principles.

Round Optimality. For our applications of interest, the communication among committee members happens over a blockchain. This makes communication costly in both financial and efficiency terms: The rate of exchanged messages is bounded by the rate at which new blocks appear, which can be in the order of minutes. For this reason we aim to minimize interaction as much as possible. Ideally, an authentication attempt should consist of a single message from the user to all committee members and of a single round of response, where the output of the protocol can be recovered without any further interaction.

Guaranteed Output Delivery. As one cannot trust all committee members, a corrupted user might decide at any point not to respond to any query. We want to guarantee that authentication protocols can still take place even if an arbitrary subset of committee members (up to some user-defined threshold) goes offline. This property is commonly known as guaranteed output delivery.

Public Verifiability. We require that the outcome of the authentication process is publicly verifiable by any external observer. This feature is needed to allow the smart contract to decide whether the user transaction should take place or not.

Resilience Against Offline Dictionary Attacks. The secrets stored by the user typically come from a low-entropy distribution (e.g., a human-memorable password or answer to some security question). For this reason, it is important the information that an attacker can gather by eavesdropping the communication does not allow it to launch brute-force attacks where the authentication process is run locally until it succeeds. Clearly the attacker can query the online authentication mechanism, but an unusual number of attempts will trigger a rate-limiting mechanism.

Resilience Against Replay Attacks. In our context authentication is tied to a specific bitstring τ, e.g., we want to make sure that an unusual transaction τ is initiated by the legitimate user. It is therefore important to ensure that previous successful authentications for a string τ cannot be mauled into successful authentications for a different string τ', without the knowledge of the secret.

5.2. Syntax and Security Properties

In the following we present the syntax for a DZT protocol. To incorporate the fact that a query is associated with some transaction trans we augment our scheme with tags τ, which we assume to be λ-bit strings. This is without loss of generality since one can set $\tau = H(\mathsf{trans})$, where $H : \{0,1\}^* \to \{0,1\}^\lambda$ is a collision resistant hash function.

Definition 4 (DZT)**.** *A DZT scheme consists of the following efficient algorithms.*

$\underline{\mathsf{Setup}(1^\lambda, t, n, \alpha)}$: *On input the security parameter 1^λ, a threshold size t, a committee size n, and a string $\alpha \in \{0,1\}^*$, the setup algorithm returns a public key pk and a vector of secret keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.*

$\underline{\mathsf{Query}(\mathsf{pk}, \beta, \tau)}$: *On input the public key pk, a string $\beta \in \{0,1\}^*$, and a tag $\tau \in \{0,1\}^\lambda$, the query algorithm returns a query q.*

$\underline{\mathsf{Response}(\mathsf{pk}, \mathsf{sk}_i, q, \tau, i)}$: *On input the public key pk, a secret key sk_i, a query q, a tag τ, and an index i, the response algorithm returns a partial response p_i.*

$\underline{\mathsf{Verdict}(\mathsf{pk}, p_1, \ldots, p_{\tilde{t}}, \tau)}$: *On input the public key pk, a set of partial responses $(p_1, \ldots, p_{\tilde{t}})$, for some $\tilde{t} \leq n$, and a tag τ, the verdict algorithm returns a bit $\{0,1\}$.*

For correctness, we require that for all $\lambda \in \mathbb{N}$, all polynomials $n = n(\lambda)$, all $t \leq n$, all sets $S \subseteq \{1, \ldots, n\}$ of size $|S| \geq t$, all strings α and τ, all $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ in the support of $\mathsf{Setup}(1^\lambda, t, n, \alpha)$ it holds that

$$\Pr\left[\mathsf{Verdict}\left(\mathsf{pk}, \{p_i\}_{i \in S}, \tau\right) = 1\right] = 1$$

where $p_i \leftarrow \mathsf{Response}(\mathsf{pk}, \mathsf{sk}_i, \mathsf{Query}(\mathsf{pk}, \alpha, \tau), \tau, i)$. Note that correctness is required to hold as long as *any* set of committee members of size at least t participates in the response to the queries. In other words, the protocol has guaranteed output delivery as long as at most $n-t$ parties are corrupted and may refuse to respond to queries or give an incorrect response.

Next we define the notion of security and we argue why it models the attackers capability in a faithful way. Our definition are inspired by those of Bellare, Pointcheval, and Rogaway [6] imported to our settings. We denote by D set of distinct strings from where the secret is chosen. Our definition captures both the cases where D is small and exponentially large, as long as one can efficiently sample an element from D.

Definition 5 (Security)**.** *A DZT is secure if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all PPT algorithms $\mathcal{A}$ it holds that*

$$\Pr\left[\mathsf{Exp}_{\mathcal{A}}(1^\lambda) = 1\right] \leq (|Q|+1)/|D| + \mathsf{negl}(\lambda)$$

where the experiment is defined in the following.

$\underline{\mathsf{Exp}_{\mathcal{A}}(1^\lambda)}$*:*

- $(t, n) \leftarrow \mathcal{A}(1^\lambda)$
- $\alpha \leftarrow\!\!\$\ D$
- $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow \mathsf{Setup}(1^\lambda, t, n, \alpha)$
- $S^* \leftarrow \mathcal{A}(\mathsf{pk})$
- $(q^*, \tau^*, p_1^*, \ldots p_{\tilde{t}}^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\{\mathsf{sk}_i\}_{i \in S^*})$

where $|S^*| = t-1$ and $\mathcal{O}$ gives the adversary access to the following interfaces:

1) $\mathsf{Accept}(\tau)$: *On input a tag τ, the adversary is given $q \leftarrow \mathsf{Query}(\mathsf{pk}, \alpha, \tau)$ and $p_i \leftarrow \mathsf{Response}(\mathsf{pk}, \mathsf{sk}_i, q, \tau, i)$, for all $i \notin S^*$.*
2) $\mathsf{Reject}(\tau, f)$: *On input a tag τ and a function f, the adversary is given $q \leftarrow \mathsf{Query}(\mathsf{pk}, f(\alpha), \tau)$ together with $p_i \leftarrow \mathsf{Response}(\mathsf{pk}, \mathsf{sk}_i, q, \tau, i)$, for all $i \notin S^*$. We require that f is given as a polynomial-size circuit and that it has no fixed point, i.e., there exists no x such that $f(x) = x$.*
3) $\mathsf{Malicious}(q, \tau)$: *On input a query q and a tag τ, the adversary is given $p_i \leftarrow \mathsf{Response}(\mathsf{pk}, \mathsf{sk}_i, q, \tau, i)$, for all $i \notin S^*$.*

We define Q to be the set of queries to the Malicious *interface and Q' be the set of queries to the* Accept *interface. The experiment returns 1 if and only if*

$$\mathsf{Verdict}(\mathsf{pk}, p_1^*, \ldots p_{\tilde{t}}^*, \tau^*) = 1 \text{ and } \tau^* \notin Q'.$$

We now discuss how the experiment defined above models the intuition for the security properties that we want to ensure. First observe that the adversary is allowed to see arbitrarily many accepting and rejecting queries from the honest users without affecting its success probability. This means that no matter how many queries it eavesdrops, its advantage in guessing α should not change. Instead, the Malicious interfaces allows the attacker to guess the value of α exactly once per query and we require that nothing is revealed beyond whether his guess is correct or not. This prevents offline attacks where the attacker locally tests for the correct value of α and returns an accepting query without querying the Malicious interface.

Finally observe that the experiment captures replay attacks: If the adversary was able to maul an honest accepting query into a query for a different tag, it could violate the above definition with a single query to the Accept interface.

6. Construction of Distributed Zero-Tester

In this section we present our DZT construction and we show that it satisfies all of the properties of interest. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2)$ be an asymmetric bilinear group of prime order p and let $H : \{0,1\}^* \to \mathbb{G}_2$ be a hash function modeled as a random oracle. We assume the existence of NIZK scheme $(\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ for NP. The scheme is described in Figure 1. For completeness, we also present a self-contained version of the underlying threshold homomorphic encryption scheme in Appendix A.

To see why the scheme is correct, note that

$$\begin{aligned}
&\left(c_0 \cdot \prod_{i=0}^{\log(p)} c_{0,i}, c_1 \cdot \prod_{i=0}^{\log(p)} c_{1,i}\right) \\
&= \left(c_0 \cdot \prod_{i=0}^{\log(p)} d_0^{2^i \cdot \alpha_i} \cdot g_1^{r_i}, c_1 \cdot \prod_{i=0}^{\log(p)} d_1^{2^i \cdot \alpha_i} \cdot h^{r_i}\right) \\
&= \left(c_0 \cdot d_0^{\alpha} \cdot g_1^{\tilde{r}}, c_1 \cdot d_1^{\alpha} \cdot h^{\tilde{r}}\right) \\
&= \left(g_1^{r} \cdot g_1^{s\alpha + \tilde{r}}, h^{r} \cdot g_1^{-\rho\alpha} \cdot g_1^{\rho\alpha} \cdot h^{s\alpha+\tilde{r}}\right) \\
&= \left(g_1^{\psi}, h^{\psi}\right)
\end{aligned}$$

where $\tilde{r} = \sum_{i=1}^{\log(p)} r_i$ and $\psi = r + s\alpha + \tilde{r}$. Then we have that

$$(b_0, b_1) = (e(g_1^{\psi}, k), e(h^{\psi}, k)) = (g_T^{\psi\kappa}, g_T^{x\psi\kappa})$$

and therefore for all admissible sets S of size $|S| = t$ we have

$$\prod_{i \in S} b_0^{x_i \lambda_i} = g_T^{x\psi\kappa} = b_1$$

$\underline{\mathsf{Setup}(1^\lambda, t, n, \alpha)}$: On input the security parameter 1^λ, a threshold size t, a committee size n, and a string $\alpha \in \mathbb{Z}_p$, the setup algorithm samples a tuple $(x, r, s, \rho) \leftarrow_\$ \mathbb{Z}_p^*$ and a t-out-of-n Shamir secret sharing $(x_1, \ldots, x_n) \leftarrow \mathsf{Share}(x, t, n)$. Then it computes $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$. The algorithm sets the public key of the scheme to

$$\mathsf{pk} = (\mathsf{crs}, h, h_1, \ldots, h_n, c_0, c_1, d_0, d_1) = (\mathsf{crs}, g_1^x, g_T^{x_1}, \ldots, g_T^{x_n}, g_1^r, h^r \cdot g_1^{-\rho\alpha}, g_1^s, h^s \cdot g_1^\rho)$$

and the secret keys to $sk_i = x_i$, for all $i \in \{1, \ldots, n\}$.

$\underline{\mathsf{Query}(\mathsf{pk}, \beta, \tau)}$: On input the public key pk, a string $\beta \in \mathbb{Z}_p$, and a tag $\tau \in \{0,1\}^\lambda$, the query algorithm does the following. For all $i \in \{0, \ldots, \log(p)\}$, sample a uniform $r_i \leftarrow_\$ \mathbb{Z}_p^*$ and compute

$$(c_{0,i}, c_{1,i}) = \left(d_0^{2^i \cdot \beta_i} \cdot g_1^{r_i}, d_1^{2^i \cdot \beta_i} \cdot h^{r_i}\right)$$

where $\beta = (\beta_1, \ldots, \beta_{\log(p)})$ is parsed in its binary representation. Then compute $\pi_i \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, \mathsf{stmt}_i, r_i)$ where

$$\mathsf{stmt}_i = \left\{\exists\ r_i \ \textit{s.t.}\ (c_{0,i}, c_{1,i}) = (g_1^{r_i}, h^{r_i})\ \textit{OR}\ (c_{0,i}, c_{1,i}) = \left(d_0^{2^i} \cdot g_1^{r_i}, d_1^{2^i} \cdot h^{r_i}\right) \middle\| \tau \right\}.$$

The algorithm returns $q = (c_{0,0}, c_{1,0}, \pi_0, \ldots, c_{0,\log(p)}, c_{1,\log(p)}, \pi_{\log(p)})$.

$\underline{\mathsf{Response}(\mathsf{pk}, \mathsf{sk}_i, q, \tau, i)}$: On input the public key pk, a secret key sk_i, a query q, a tag τ, and an index i, the response algorithm checks whether for all $i \in \{0, \ldots, \log(p)\}$ it holds that

$$\mathsf{NIZK.Verify}(\mathsf{crs}, \mathsf{stmt}_i, \pi_i) = 1$$

and aborts if this condition is not verified. Then the algorithm evaluates $k \leftarrow H(\mathsf{pk}, q, \tau)$ and computes

$$b_0 = e\left(c_0 \cdot \prod_{i=0}^{\log(p)} c_{0,i}, k\right) \text{ and } b_1 = e\left(c_1 \cdot \prod_{i=0}^{\log(p)} c_{1,i}, k\right).$$

Furthermore, the algorithm computes $b^{(i)} = b_0^{x_i}$ and an equality proof $\tilde{\pi} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, \tilde{\mathsf{stmt}}_i, x_i)$ where

$$\tilde{\mathsf{stmt}}_i = \left\{\exists\ x_i \ \textit{s.t.}\ b^{(i)} = b_0^{x_i}\ \textit{AND}\ h_i = g_T^{x_i} \middle\| \tau \right\}.$$

The algorithm returns $p_i = (b^{(i)}, b_1, \tilde{\pi}, \tau, i)$.

$\underline{\mathsf{Verdict}(\mathsf{pk}, p_1, \ldots, p_{\bar{t}}, \tau)}$: On input the public key pk, a set of partial responses $(p_1, \ldots, p_{\bar{t}})$, and a tag τ, the verdict algorithm checks whether there exists a set S of responses of size $|S| = t$ such that the corresponding values (b_1, τ) are identical for all responses and

$$\mathsf{NIZK.Verify}(\mathsf{crs}, \tilde{\mathsf{stmt}}_i, \tilde{\pi}_i) = 1$$

for all $i \in S$. If this is the case, the algorithm checks whether

$$\prod_{i \in S} \left(b^{(i)}\right)^{\lambda_i} = b_1$$

and returns 1 if and only if all of the above conditions are satisfied.

Figure 1. Our DZT protocol.

with probability 1. We now show that our DZT scheme satisfies the notion of security.

Theorem 1 (Security)**.** *If the XDH and the DBDH problems are hard over $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2)$ then the DZT construction as described in Figure 1 is secure.*

Proof. We assume without loss of generality that the adversary outputs a maximally corrupted subset of parties S^* of size $|S| = t - 1$. Then we gradually modify the experiment in the following sequence of hybrids.

Hybrid 0: This is the original experiment $\mathsf{Exp}_{\mathcal{A}}(1^\lambda)$.

Hybrid 1: In this hybrid all NIZK proof issued to the adversary on behalf of honest parties are computed using the simulator instead of the real witness. This modification is computationally indistinguishable by a standard hybrid argument against the zero-knowledge property of the NIZK system.

Hybrid 2: In this hybrid all NIZK proof sent by the adversary are extracted using the Ext algorithm, provided by the simulation extractability property of the NIZK system. If any of the outputs of the extractor does not constitute a valid witness for the corresponding relation, then the experiment aborts. A standard hybrid argument can be used to show that the probability that an abort is triggered is bounded by a negligible function in the security parameter, by the simulation extractability of the NIZK system.

Hybrid 3: In this hybrid we change how the responses of the honest parties are computed for all queries of the adversary. Fix a query of the adversary (to any of the

interfaces) and let (b_0, b_1) be defined as in the Response algorithm. The output of the i-th honest party is identical to the original experiment except for

$$b^{(i)} = \left(\frac{b_1}{\prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} \cdot g_T^{\kappa\rho \cdot (\tilde{\alpha} - \alpha)}} \right)^{\lambda_i^{-1}}.$$

where $k = g_2^{\kappa}$, and $\tilde{\alpha}$ depends on the type query that the adversary is asking:

(1) For queries to the Accept interface, $\tilde{\alpha}$ is set to α.
(2) For queries to the Reject interface, $\tilde{\alpha}$ is set to $f(\alpha)$.
(3) For queries to the Malicious interface, the values of $(\beta_0, \ldots, \beta_{\log(p)})$ are read from the extracted NIZK sent by the adversary, and the value of $\tilde{\alpha}$ is set to $\sum_{i=0}^{\log(p)} 2^i \cdot \beta_i$.

Observe that such a response is correctly distributed since

$$\left(b^{(i)}\right)^{\lambda_i} \cdot \prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} = \frac{b_1}{g_T^{\kappa\rho \cdot (\tilde{\alpha} - \alpha)}} = \frac{h^{\tilde{r}} \cdot g_T^{\kappa\rho \cdot (\tilde{\alpha} - \alpha)}}{g_T^{\kappa\rho \cdot (\tilde{\alpha} - \alpha)}} = h^{\tilde{r}}$$

for some adversarially chosen $\tilde{r}$. It follows that this modification is only syntactical and the view of the adversary is identical to that of the previous hybrid.

Hybrid 4: In this hybrid the shares x_i, for all $i \in S^*$, are sampled uniformly from $\mathbb{Z}_p$, instead of being computed using the Share algorithm. Then the element h_i corresponding to the i-th honest party is computed as

$$h_i = \left(\frac{h}{\prod_{i \in S^*} h_j^{\lambda_j}} \right)^{\lambda_i^{-1}}.$$

Since $|S^*| < t$ it follows that the view of the adversary is identical to that induced by the previous hybrid.

Hybrid 5: In this hybrid we compute the ciphertexts (c_0, c_1) and (d_0, d_1) as encryptions of zero, i.e.,

$$(c_0, c_1) = (g_1^r, h^r) \text{ and } (d_0, d_1) = (g_1^s, h^s)$$

where $(r, s) \leftarrow_{\$} \mathbb{Z}_p^*$. Indistinguishability follows from two invocations of the XDH assumption.

Hybrid 6: This hybrid is identical to the previous one, except that the responses of the honest parties for all adversarial queries are computed as

$$b^{(i)} = \left(\frac{b_1}{\prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} \cdot g_T^{\gamma \cdot (\tilde{\alpha} - \alpha)}} \right)^{\lambda_i^{-1}}.$$

where $\gamma \leftarrow_{\$} \mathbb{Z}_p^*$ is sampled freshly for each adversarial query and $\tilde{\alpha}$ is defined as before.

Let Q be the set of queries issued by the adversary to the decryption oracle. For all $i \in \{1, \ldots, |Q|\}$ we define the following hybrid distribution

$$\left(g_2^{\kappa_1}, \ldots, g_2^{\kappa_Q}, g_T^{\rho\kappa_1}, \ldots, g_T^{\rho\kappa_{i-1}}, g_T^{\rho\kappa_i}, g_T^{\gamma_{i+1}} \ldots, g_T^{\gamma_Q}\right).$$

Note that on the one extreme we have that the distribution corresponds to the computation done in in Hybrid 5

$$\left(g_2^{\kappa_1}, \ldots, g_2^{\kappa_Q}, g_T^{\rho\kappa_1}, \ldots, g_T^{\rho\kappa_Q}\right)$$

whereas on the other extreme the distribution is identical to the computation done in Hybrid 6

$$\left(g_2^{\kappa_1}, \ldots, g_2^{\kappa_Q}, g_T^{\gamma_1}, \ldots, g_T^{\gamma_Q}\right).$$

One can easily show that the distance between the i-th and the $(i+1)$-th hybrids is negligible by an invocation of the the DBDH assumption: Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, g_2^x, g_T^y, g_T^z)$ be the tuple taken as input by the distinguisher. The reduction sets $g_T^{\rho} = g_T^y$ and answers the queries $\{1, \ldots, i-1\}$ as specified in Hybrid 5 and the queries $\{i+1, \ldots, |Q|\}$ as specified in Hybrid 6, programming the random oracle to some value with know discrete logarithm κ. For the i-th query the reduction programs the random oracle to $k = g_2^x$ and computes

$$b^{(i)} = \left(\frac{b_1}{\prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} \cdot g_T^{z \cdot (\tilde{\alpha} - \alpha)}} \right)^{\lambda_i^{-1}}.$$

Observe that if $z = xy$ then the view of the adversary is identical to that of the i-th hybrid distribution, whereas if z is uniform in $\mathbb{Z}_p^*$ then the view of the adversary is identical to that of the $(i+1)$-th hybrid. This shows that the hybrid distributions must be computationally indistinguishable.

Hybrid 7: In this hybrid we change the response of the honest parties to all adversarial queries. Specifically we compute

$$b^{(i)} = \left(\frac{b_1}{\prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} \cdot g_T^{\gamma \cdot \tilde{b}}} \right)^{\lambda_i^{-1}}.$$

where the bit $\tilde{b}$ is defined as follows:

(1) For queries to the Accept interface, we set $\tilde{b} = 0$.
(2) For queries to the Reject interface, we set $\tilde{b} = 1$.
(3) For queries to the Malicious interface, we set $\tilde{b} = 0$ if and only if $\alpha = \sum_{i=0}^{\log(p)} 2^i \cdot \beta_i$, where $(\beta_0, \ldots, \beta_{\log(p)})$ is extracted from the NIZK produced by the adversary.

First observe that if $\tilde{\alpha} = \alpha$, then the distribution is indentical as $\tilde{b} = \tilde{\alpha} - \alpha = 0$. Furthermore, observe that for all constants $c \neq 0$ and $c' \neq 0$ the following distributions

$$(c, c', c \cdot \gamma) \equiv (c, c', c' \cdot \gamma)$$

where $\gamma \leftarrow_{\$} \mathbb{Z}_p^*$, are identical. It follows that the view of the adversary is unchanged.

Analysis: We are now in the position of analyzing the success probability of the adversary in Hybrid 7. First observe that the public parameters of the scheme do not contain any information about α. Furthermore, the responses of the queries to the Accept and Reject interfaces are computed independently of α. On the other hand, each query to the Malicious interface reveals a single bit of information, i.e., whether $\tilde{\alpha} = \alpha$, for some adversarially chosen $\tilde{\alpha}$. Let Q be the set of queries to the Malicious interface, we can then bound the probability that the adversary guesses the correct α by $(|Q| + 1)/|D|$.

It remains to analyze the success probability that the adversary has without guessing a correct α, which is equivalent to the probability of producing a false proof for a (fresh) rejecting statement and can be bounded

to a negligible value by the simulation extractability of the NIZK system. Thus, the success probability of the adversary against Hybrid 7 is bounded from above by $(|Q| + 1)/|D| + \mathsf{negl}(\lambda)$. By the above analysis, the same bound holds for an adversary playing against the original experiment, up to a negligible additive factor in the security parameter. □

6.1. Efficient Non-Interactive Zero-Knowledge

In the following we discuss an efficient instantiation for the NIZK system, which will allow us to scale the efficiency of our system to the regime of practicality. First observe that our NIZK system must support proofs for discrete logarithm equality. This class of NIZKs can be instantiated very efficiently with the classical protocol from Schnorr [35], which we recall in the following. Let (g, h) be a pair of elements of a group of prime order p and let $(\tilde{g}, \tilde{h})$ be two group elements. We want to prove the existence of some $w \in \mathbb{Z}_p$ such that $g^w = \tilde{g}$ and $h^w = \tilde{h}$. The prover samples a uniform $r \leftarrow_\$ \mathbb{Z}_p$ and computes the commitment (g^r, h^r), then the challenge $c \in \mathbb{Z}_p$ is computed by hashing the commitment together with the statement. In our case the statement includes also the string τ, that is concatenated to the group elements and hashed to compute the challenge c. The prover computes $z = wc + r$ and returns $\pi = (g^r, h^r, z)$. The proof is considered valid if $\tilde{g}^c \cdot g^r = g^z$ and $\tilde{h}^c \cdot h^r = h^z$. It is well known [34] that the protocol is zero-knowledge and simulation sound[2] in the random oracle model.

The above protocol can be extended to handle disjunctions (i.e., the OR-composition of proofs) using a standard trick: For the case of two statements, the two proofs are computed in parallel using the same algorithm as above, except that the prover is allowed to choose the challenges c_0 and c_1 under the constraint that $c_0 + c_1 = c$ where c is computed hashing both statements and the corresponding commitments. The proof is considered valid if both of the resulting proofs correctly verify.

Finally we remark that Schnorr's NIZKs have a so called *transparent* setup which is not required to be generated by a trusted party, i.e., the setup consists of sampling the random oracle.

6.2. Implementation in Ethereum

Implementing our construction as a smart contract in Ethereum entails a new set of challenges: While in principle bilinear group operations and pairings are efficient enough to be performed on commodity machines, the integration in Ethereum of our DZT requires one to implement them in Solidity, the language of Ethereum smart contracts. Such a language imposes a hard limit on the amount of computation that each smart contract can perform, which makes implementation of cryptographic operations particularly challenging. Fortunately, Solidity has precompiled instructions for the following (bilinear) group operations:

1) Group operations in $\mathbb{G}_1$.
2) Modular exponentiations in $\mathbb{G}_1$.
3) Checks of pairing product equations $\mathsf{PPEq} : \mathbb{G}_1^n \times \mathbb{G}_2^n \to \{0, 1\}$ of the form

$$e(x_1, y_1) \cdots e(x_n, y_n) \stackrel{?}{=} 1.$$

However this semantic turns out to be insufficient to implement our scheme. The reason is that our smart contract (whose computational load consists mainly of calls to the Verdict algorithm in Figure 1) requires us to compute group operations in the target group $\mathbb{G}_T$, which are not natively supported by Solidity. One solution to circumvent this issue could be to implement group operations in $\mathbb{G}_T$ from scratch, without leveraging precompiled instructions. Unfortunately elliptic curve operations in the target group are notoriously expensive, and this approach causes the smart contract to exceed the maximum amount of computation allowed by the specifications of Ethereum.

To circumvent this issue, we modify our scheme to be compatible with precompiled instructions in Solidity. Our solution is sketched in what follows. Recall that in our scheme each committee member provides the smart contract with a target group element

$$b^{(i)} = e(\bar{c}, k)^{\mathsf{sk}_i}$$

(along with other information that is irrelevant for this overview). In our modified scheme, each member publishes instead

$$\bar{c}^{\mathsf{sk}_i \cdot \psi} \text{ and } k^{1/\psi},$$

where ψ is a uniformly sampled integer in $\mathbb{Z}_p^*$. Note that we can now compute the Verdict algorithm with a precompiled instruction PPEq (3) and the newly introduced term ψ does not affect the correctness of the scheme, since it cancels out with the pairing. To see why this is the case, observe that

$$e\left(\bar{c}^{\mathsf{sk}_i \cdot \psi}, k^{1/\psi}\right) = e(\bar{c}, k)^{\mathsf{sk}_i} = b^{(i)}.$$

We stress that the presence of the randomness ψ is crucial to avoid mix-and-match attacks. We proceed by presenting our modified scheme more formally.

Construction. We present our modified DZT construction. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2)$ be an asymmetric bilinear group of prime order p and let $H : \{0,1\}^* \to \mathbb{G}_2$ be a hash function modeled as a random oracle. We assume the existence of a NIZK scheme $(\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ for NP. The scheme is described in Figure 2. Correctness follows by a routine calculation.

Note that the modified statement that we need to prove in the response algorithm is again a proof of discrete logarithm equality, except that the first equation involves the comparison of target group elements, which is inefficient. To circumvent this issue, we run the Schnorr NIZK over the source group $\mathbb{G}_1$ and we augment the verification with a pairing. More specifically, we let the prover sample a uniform $r \leftarrow_\$ \mathbb{Z}_p$ and compute the commitment $(\tilde{c}_0^r, g_1^r)$. The challenge c is obtained by hashing the commiment and the statement, then the prover computes $z = x_i \cdot c + r$. The proof consists of $(\tilde{c}_0^r, g_1^r, z)$. The verifier accepts if

$$e\left(\left(c_0^{(i)}\right)^c, k^{(i)}\right) \cdot e\left(\tilde{c}_0^r, k\right) = e\left(\tilde{c}_0^z, k\right)$$

and

$$h_i^c \cdot g_1^r = g_1^z$$

2. While our DZT construction requires the stronger notion of simulation extractable NIZK, we settle for simulation soundness in favor of a more efficient scheme, as commonly done in the literature.

$\underline{\mathsf{Setup}(1^\lambda, t, n, \alpha)}$: On input the security parameter 1^λ, a threshold size t, a committee size n, and a string $\alpha \in \mathbb{Z}_p$, the setup algorithm samples a tuple $(x, r, s, \rho) \leftarrow_\$ \mathbb{Z}_p^*$ and a t-out-of-n Shamir secret sharing $(x_1, \ldots, x_n) \leftarrow \mathsf{Share}(x, t, n)$. Then it computes $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$. The algorithm sets the public key of the scheme to

$$\mathsf{pk} = (\mathsf{crs}, h, h_1, \ldots, h_n, c_0, c_1, d_0, d_1) = (\mathsf{crs}, g_1^x, g_1^{x_1}, \ldots, g_1^{x_n}, g_1^r, h^r \cdot g_1^{-\rho\alpha}, g_1^s, h^s \cdot g_1^\rho)$$

and the secret keys to $sk_i = x_i$, for all $i \in \{1, \ldots, n\}$.

$\underline{\mathsf{Query}(\mathsf{pk}, \beta, \tau)}$: Same as Figure 1.

$\underline{\mathsf{Response}(\mathsf{pk}, \mathsf{sk}_i, q, \tau, i)}$: On input the public key pk, a secret key sk_i, a query q, a tag τ, and an index i, the response algorithm checks whether for all $i \in \{0, \ldots, \log(p)\}$ it holds that

$$\mathsf{NIZK.Verify}(\mathsf{crs}, \mathsf{stmt}_i, \pi_i) = 1$$

and aborts if this condition is not verified. Then the algorithm evaluates $k \leftarrow H(\mathsf{pk}, q, \tau)$ and computes

$$\tilde{c}_0 = c_0 \cdot \prod_{i=0}^{\log(p)} c_{0,i} \text{ and } \tilde{c}_1 = c_1 \cdot \prod_{i=0}^{\log(p)} c_{1,i}.$$

Furthermore, the algorithm samples some $\psi \leftarrow_\$ \mathbb{Z}_p^*$ and sets $c_0^{(i)} = \tilde{c}_0^{\psi \cdot x_i}$ and $k^{(i)} = k^{1/\psi}$. Then it computes an equality proof $\tilde{\pi} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, \tilde{\mathsf{stmt}}_i, x_i)$ where

$$\tilde{\mathsf{stmt}}_i = \left\{ \exists\ x_i \ s.t.\ e\left(c_0^{(i)}, k^{(i)}\right) = e\left(\tilde{c}_0, k\right)^{x_i} \ \textit{AND}\ h_i = g_1^{x_i} \middle\| \tau \right\}.$$

The algorithm returns $p_i = \left(c_0^{(i)}, \tilde{c}_0, \tilde{c}_1, k^{(i)}, k, \tilde{\pi}, \tau, i\right)$.

$\underline{\mathsf{Verdict}(\mathsf{pk}, p_1, \ldots, p_{\tilde{t}}, \tau)}$: On input the public key pk, a set of partial responses $(p_1, \ldots, p_{\tilde{t}})$, and a tag τ, the verdict algorithm checks whether there exists a set S of responses of size $|S| = t$ such that the corresponding values $(\tilde{c}_0, \tilde{c}_1, k, \tau)$ are identical for all responses and

$$\mathsf{NIZK.Verify}(\mathsf{crs}, \tilde{\mathsf{stmt}}_i, \tilde{\pi}_i) = 1$$

for all $i \in S$. If this is the case, the algorithm checks whether

$$\prod_{i \in S} e\left(\left(c_0^{(i)}\right)^{\lambda_i}, k^{(i)}\right) = e\left(\tilde{c}_1, k\right)$$

and returns 1 if and only if all of the above conditions are satisfied.

Figure 2. Ethereum-compatible DZT protocol.

where the challenge c is recomputed locally by the verifier. Since such a protocol is an instance of Schnorr NIZK, zero-knowledge and simulation extractability are immediate.

Analysis. We now analyze the security of our new scheme. One downside of this modification is that its security relies on a new (static) assumption over asymmetric bilinear groups that we introduce in this work, which we refer to as the *dual* Diffie-Hellman (dDH) assumption. As a positive evidence that the problem is likely to be intractable, we show that the assumption holds in the generic group model [37] in Appendix B.

Assumption 3 (dDH)**.** *Let $\mathcal{G}$ be a bilinear group generator. $\mathcal{G}$ is dDH-hard if for all PPT distinguishers it holds that the following distributions are computationally indistinguishable*

$$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, g_1^x, g_1^y, g_1^z, g_1^{zy/x}, g_2^v, g_2^{xv}, g_2^w, g_2^{yw}) \approx (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, g_1^x, g_1^y, g_1^z, g_1^u, g_2^v, g_2^{xv}, g_2^w, g_2^{yw})$$

where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2) \leftarrow_\$ \mathcal{G}(1^\lambda)$ *and* $(u, v, w, x, y, z) \leftarrow_\$ \mathbb{Z}_p^*$.

We are now ready to show that the scheme satisfies the security notion for a DZT, assuming the hardness of the XDH and the dDH problems. Due to space constraints, the proof is deferred to Appendix B.

Theorem 2 (Security)**.** *If the XDH and the dDH problems are hard over $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2)$ then the DZT construction as described in Figure 2 is secure.*

7. Experimental Evaluations

We implement the DZT-based solution as a smart contract on the Ethereum blockchain. The smart contract is written in Solidity, an Ethereum specific language. Contracts written in Solidity are compiled into bytecode for the Ethereum Virtual Machine (EVM). The code is available at [1]. The runtime of programs on the EVM is limited by a unit called gas. Every operation in the EVM costs gas and there is a hard limit as to how much gas a transaction can consume. This gives a hard limit on how much computation can be done in one transaction. Since the price of Ethereum and gas is fluctuating, all costs displayed in Dollars are calculated with a fixed conversion rate of 1 gas costing 1 GWei and 1 Ether costing 180$.

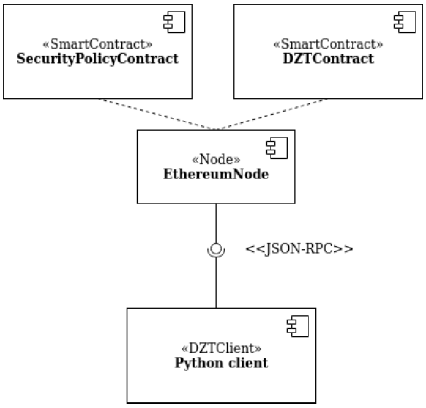

Figure 3. System architecture of our DZT implementation

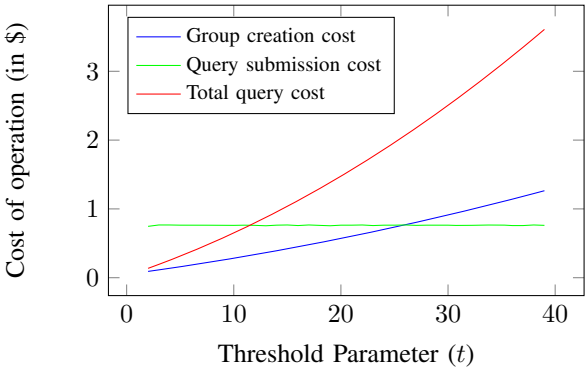

Figure 4. Cost of the creation of a DZT group (blue) cost for a client querying the group (green) and cost of submitting and verifying a query (red).

The non-blockchain parts of the implementations use the JSON-RPC API of an Ethereum node to interact with the smart contracts. A pictorial description of the system architecture is given in Figure 3.

7.1. Security Policies

The security policy of a smart contract determines when a transaction triggers the usage of a second factor for authentication. In our implementation, the smart contract tracks the total flow of money and the amounts transferred to specific addresses. This allows us to set fine-grained security policies that depends on the history of transactions. We say that a transaction is flagged if the security policy invokes the 2FA protocol for such a transaction. In our prototype, we consider four types of security policies:

(1) Flag every transaction to any new beneficiary.
(2) Flag a transaction if the sum of all coins transferred (since the last flag) is above a predefined threshold.
(3) Flag a transaction if the sum of all funds that are transferred in a specific timeframe (a sliding window) exceeds a predefined threshold.
(4) Flag a transaction if the sum of all coins sent to a specific beneficiary (since the last flag) exceeds a predefined threshold.

This list of policies is non-exclusive and multiple policies can coexist in the same smart contract. If any security policy flags a transaction, the counter for such a policy is reset if and only if the transaction is successfully authenticated.

We measured the overhead that the implementation of each policy adds to processing transactions. For policy (1), it cost 0.005$ for a user to set it up. If a transaction is flagged because it is sent to a therefore unknown beneficiary, the user has to pay 0.008$ to now save that the beneficiary is verified for further transactions. For policies (2) and (4), the setup cost is 0.009$ and each transaction costs an additional 0.006$ to sum up the coins transferred. If the threshold is reached, the user has to spend 0.004$ (and 0.006$, respectively) to reset the sum of all coins spent to 0. The cost of policy (3) varies with the size of the sliding window, which we measure for values ranging from 0 to 100 transactions. The added cost does not exceed 0.5$, in a non-optimized implementation.

7.2. Performance Evaluation

The steps of the 2FA mechanism are implemented as functions on a smart contract. The smart contract notifies the committee members via Ethereum events if a new protocol step has to be executed, i.e., this happens if a new query is submitted. The contract then runs the Verdict algorithm, which internally verifies the NIZKs and checks whether the conditions dictated by the algorithm are all verified. We stress that the worst-case runtime of the Verdict algorithm is only proportional to $\tilde{t}$ (the number of partial responses fed as input). Recall that the Verdict algorithm consist of two subroutines:

1) Checking the NIZKs: There are at most $\tilde{t}$ NIZKs, so this step can be done in time linear in $\tilde{t}$.
2) Partition the responses in sets with the same value of (b_1, τ): This operation can also be done in time proportional to $\tilde{t}$.

At this point we can check whether there exists a set with more than t elements where all NIZKs verify. Note that, within such a set, *any* subset of the shares in this set is considered valid (since all the NIZKs verify correctly), so the Verdict algorithm can just pick an arbitrary one (e.g. the lexicographically smallest one).

We use the 256bit version of the Barreto-Naehrig (BN) curves to instantiate the pairings [5]. This curve is especially suited for use in Ethereum, since an elements of its base field fits into the 256bit wide registers of the EVM. We show the costs associated with all operations in Figure 4. For our experiments, we set the threshold size (t) to be equal to the group size (n), since decreasing the value of t only decreases the costs of each algorithm. For the client, the cost of the setup phase grows linearly with the size of the committee. The cost of the queries depends only on the threshold parameter t, i.e., the maximum size of parties that the attacker can corrupt. The cost to verify a query is identical for all members of the committee and we only display the cumulative cost (i.e. the sum of the costs for each committee member), which grows linearly with t. We stress that a lower value of t implies a lower corruption threshold but at the same time an increase in reliability, as only t parties need to be online in order to run protocol.

8. A Solution Based on U2F Tokens

The Universal Second Factor (U2F) is a token-based authentication protocol specified by the FIDO Alliance [38]. It is used to enhance the security of standard password-based authentication with a cryptographic secret stored on a hardware token. This protocol is supported by all major browser and by many web services (such as Gmail, Facebook, and Dropbox). The U2F protocol consists of three interacting entities: (1) A hardware token, (2) a client, and (3) a relying party, which guards access to the resources. The U2F protocol specifies two subroutines, a one-time *registration* and and unbounded number of *authentications*, which we briefly describe below.

Registration. The registration allows the relying party to bind a physical token to a user account. The relying party is associated with some application identifier (e.g., the url of its website) and begins the registration phase by sending a challenge to the token. The hardware token generates a fresh ECDSA key pair $(\mathsf{sk}, \mathsf{vk})$ and sends back the verification key and a *key-handle* to the relying party. At this point the relying party associates the key handle and vk with the account of the user.

Authentication. Upon each authentication request, the relying party sends the key handle, the identifier, and a challenge to the token. The key handle is used by the token to recover the corresponding signing key, which is used to sign the challenge of the relying party. In addition to the challenge, the token also signs a counter, which is maintained by the token locally and increased upon each authentication run. The relaying party considers the authentication successful if the signature correctly verifies against the verification key vk associated with the user account and if the counter (also included in the signed message) increased from the previous authentication.

8.1. U2F on Ethereum

In the U2F token-based authentication protocol, the smart contract impersonates the relying party. The authentication protocol for a transaction τ begins with the client querying the smart contract for a challenge and receives a random nonce and a key handle that is used by the token to specify the correct key to use. The client queries the token for a signature on the nonce and on τ and sends it to the smart contract, which can publicly verify the validity of the signature. A client can use the same token with multiple smart contracts, since U2F tokens allow one to generate an arbitrary number of key pairs, each associated to a unique identifier. In our case, the identifier consists of the address of the smart contract and the chain identity.

Nonce Sampling. A subtlety that needs to be addresses is that Ethereum smart contracts do not support non-deterministic operation and therefore cannot sample truly random nonces. We circumvent this limitation by setting the nonce to $H(\tau)$, where H is a hash function modeled as a random oracle and τ is a unique identifier of the transaction. In Ethereum, a transaction can be uniquely identified by the concatenation of the receiver and sender addresses and the value of a counter, which denotes the number of the transaction from the sending address and can only grow over time.

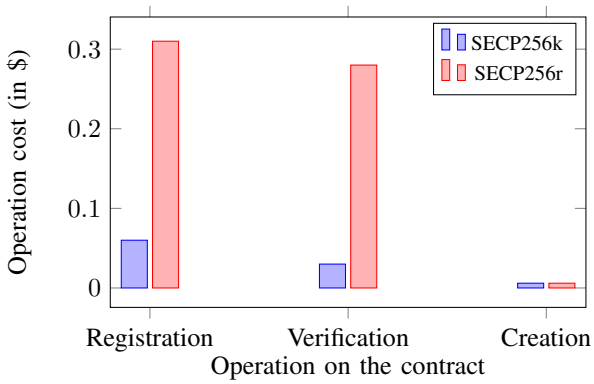

Figure 5. Transaction registration, verification and creation costs on curve SECP256k (blue) and curve SECP256r (red).

Implementation. U2F specifies curve SECP256r1 or P-256 as the basis for signature creation and verification [19]. On the other hand, Ethereum uses signatures over the curve SECP256k1 internally and offers an EVM instruction to access the signature verification capabilities that is part of the official standard [39]. To make the two choices compatible, we design the smart contract to allow a user to decide at registration time which curve should be used for verification.

The costs for authenticating transactions are shown in Figure 5. All measurements are taken using a simulated token. The use of a soft token does not impact the messages that are exchanged on the blockchain and the interaction between the smart contract and the user client, so this change does not affect the costs. The measurements are taken as the average over 100 transactions. The cost of verifying signatures on the standard curve SECP256r of U2F is significantly higher compared to verification on curve SECP256k: Standard U2F tokens have a one-time registration fee of 0.31$ and a recurring fee of 0.28$ for verifying transaction requests. For the costs over the SECP256k curve, the one-time registration costs 0.07$ and the recurring transaction verification costs 0.03$. The evaluation shows that the usage of U2F in the blockchain setting is feasible in terms of costs. There is a big cost benefit in switching the standard U2F signature curve to one which can be natively verified by a smart contract, however this also requires custom made tokens.

9. Conclusions

In this work we proposed a new system to safeguard users' transactions, where a smart contract is entitled to request for a two-factor authentication, in case of exceptional events. We designed and implemented two 2FA protocols in Ethereum. Our performance evaluation shows that augmenting Ethereum with 2FA comes at minimal additional cost.

Acknowledgements. Supported in part by the NSF award 1916939, a gift from Ripple, a DoE NETL award, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, a Cylab seed funding award, the InterACT network, and a Baden-Württemberg scholarship.

References

[1] https://github.com/FBreuer2/security-policies-in-crypto.

[2] "How to steal $500 million in cryptocurrency," https://fortune.com/2018/01/31/coincheck-hack-how/.

[3] "Parity technologies," https://www.parity.io/security-alert-2/.

[4] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu, "Password-protected secret sharing," in *Proceedings of the 18th ACM conference on Computer and Communications Security*, 2011, pp. 433–444.

[5] P. S. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *International Workshop on Selected Areas in Cryptography*. Springer, 2005, pp. 319–331.

[6] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2000, pp. 139–155.

[7] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," 1988.

[8] D. Boneh and X. Boyen, "Secure identity based encryption without random oracles," in *Annual International Cryptology Conference*. Springer, 2004, pp. 443–459.

[9] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *Annual international cryptology conference*. Springer, 2004, pp. 41–55.

[10] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Annual international cryptology conference*. Springer, 2001, pp. 213–229.

[11] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. Rasmussen, and A. Sahai, "Threshold cryptosystems from threshold fully homomorphic encryption," in *Annual International Cryptology Conference*. Springer, 2018, pp. 565–596.

[12] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 337–367.

[13] R. Canetti and S. Goldwasser, "An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 90–106.

[14] C. Decker and R. Wattenhofer, "Bitcoin transaction malleability and mtgox," in *European Symposium on Research in Computer Security*. Springer, 2014, pp. 313–326.

[15] M. Di Raimondo and R. Gennaro, "Provably secure threshold password-authenticated key exchange," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2003, pp. 507–523.

[16] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Secure two-party threshold ecdsa from ecdsa assumptions," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 980–997.

[17] ——, "Threshold ecdsa from ecdsa assumptions: The multiparty case," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1051–1066.

[18] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[19] S. for Efficient Cryptography Group, "Sec 2: Recommended elliptic curve domain parameters," 10 2019. [Online]. Available: https://www.secg.org/sec2-v2.pdf

[20] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ecdsa with fast trustless setup," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 1179–1194.

[21] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security," in *International Conference on Applied Cryptography and Network Security*. Springer, 2016, pp. 156–174.

[22] C. Gentry *et al.*, "Fully homomorphic encryption using ideal lattices." in *Stoc*, vol. 9, no. 2009, 2009, pp. 169–178.

[23] N. Gilboa and Y. Ishai, "Distributed point functions and their applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2014, pp. 640–658.

[24] S. Goldfeder, R. Gennaro, H. Kalodner, J. Bonneau, J. A. Kroll, E. W. Felten, and A. Narayanan, "Securing bitcoin wallets via a new dsa/ecdsa threshold signature scheme," 2015.

[25] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of computer and system sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[26] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu, "Toppss: cost-minimal password-protected secret sharing based on threshold oprf," in *International Conference on Applied Cryptography and Network Security*. Springer, 2017, pp. 39–58.

[27] A. Joux, "A one round protocol for tripartite diffie–hellman," in *International algorithmic number theory symposium*. Springer, 2000, pp. 385–393.

[28] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *annual international conference on the theory and applications of cryptographic techniques*. Springer, 2008, pp. 146–162.

[29] Y. Lindell, "Fast secure two-party ecdsa signing," in *Annual International Cryptology Conference*. Springer, 2017, pp. 613–644.

[30] Y. Lindell and A. Nof, "Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1837–1854.

[31] P. MacKenzie, T. Shrimpton, and M. Jakobsson, "Threshold password-authenticated key exchange," in *Annual International Cryptology Conference*. Springer, 2002, pp. 385–400.

[32] M. I. Mehar, C. L. Shier, A. Giambattista, E. Gong, G. Fletcher, R. Sanayhie, H. M. Kim, and M. Laskowski, "Understanding a revolutionary and flawed grand experiment in blockchain: the dao attack," *Journal of Cases on Information Technology (JCIT)*, vol. 21, no. 1, pp. 19–32, 2019.

[33] M. Möser, I. Eyal, and E. G. Sirer, "Bitcoin covenants," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 126–141.

[34] D. Pointcheval and J. Stern, "Security proofs for signature schemes," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1996, pp. 387–398.

[35] C.-P. Schnorr, "Efficient signature generation by smart cards," *Journal of cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[36] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[37] V. Shoup, "Lower bounds for discrete logarithms and related problems," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1997, pp. 256–266.

[38] S. Srinivas, D. Balfanz, E. Tiffany, A. Czeskis, and F. Alliance, "Universal 2nd factor (u2f) overview," *FIDO Alliance Proposed Standard*, pp. 1–5, 2015.

[39] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger."

Appendix

1. Threshold Homomorphic Encryption from Bilinear Maps

In the following we present our threshold homomorphic encryption scheme for linear predicates from bilinear maps.

Definition. A threshold homomorphic encryption (THE) allows one to share the keys for a homomorphic encryption scheme to a set of n parties such that any set that satisfies the corresponding access structure can reconstruct

the evaluation of some homomorphic computation. We recall the definition from [11].

Definition 6 (THE)**.** *A THE scheme for a function family $\mathcal{F}$ and n parties consists of the following efficient algorithms.*

$\underline{\mathsf{Setup}(1^\lambda, \mathbb{A})}$: *On input the security parameter 1^λ and an access structure $\mathbb{A}$, the setup algorithm returns a public key pk and a set of secret keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.*

$\underline{\mathsf{Enc}(\mathsf{pk}, m)}$: *On input the public key pk and a message m, the encryption algorithm returns a ciphertext c.*

$\underline{\mathsf{Eval}(\mathsf{pk}, f, (c_1, \ldots, c_\ell))}$: *On input the public key pk, an ℓ-input function f, and a set of ciphertexts $(c_1, \ldots, c_\ell)$, the evaluation algorithm returns an evaluated ciphertext c.*

$\underline{\mathsf{PDec}(\mathsf{sk}, c, i)}$: *On input a secret key sk, a ciphertext c, and an index i, the partial decryption algorithm returns a decryption share p.*

$\underline{\mathsf{Rec}(p_1, \ldots, p_{\tilde{t}})}$: *On input a set of shares $(p_1, \ldots, p_{\tilde{t}})$, for some $\tilde{t} \leq n$, the reconstruction algorithm returns a message m.*

We say that a scheme is compact if the size of evaluated ciphertexts does not depend on the circuit representation of the evaluated function f. Correctness requires that for all $\lambda \in \mathbb{N}$, all access structures $\mathbb{A}$, all accepting sets $S \in \mathbb{A}$, all ℓ-input functions $f \in F$, all message vectors $(m_1, \ldots, m_\ell)$, all $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ in the support of $\mathsf{Setup}(1^\lambda, \mathbb{A})$ and c_i in the support of $\mathsf{Enc}(\mathsf{pk}, m_i)$ it holds that

$$\Pr\left[\begin{array}{r}\mathsf{Rec}\left(\{\mathsf{PDec}(\mathsf{sk}_i, \mathsf{Eval}(\mathsf{pk}, f, (c_1, \ldots, c_\ell)), i)\}_{i \in S}\right) \\ = f(m_1, \ldots, m_\ell)\end{array}\right] = 1.$$

We recall the standard notion of semantic security for public-key encryption [25].

Definition 7 (Semantic Security)**.** *A THE is semantically secure if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, all access structures $\mathbb{A}$, and all PPT algorithms $\mathcal{A}$ it holds that*

$$\Pr\left[\mathcal{A}(c) = b \;\middle|\; \begin{array}{l}(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow \mathsf{Setup}(1^\lambda, \mathbb{A}) \\ (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}) \\ b \leftarrow_\$ \{0,1\} \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)\end{array}\right] = 1/2 + \mathsf{negl}(\lambda).$$

Finally we recall the notion of simulation security which, roughly speaking, says that the decryption shares should not reveal anything beyond the output of the evaluated function.

Definition 8 (Simulation Security)**.** *A THE is simulation secure if there exists a PPT simulator $(\mathsf{Sim}_0, \mathsf{Sim}_1)$ and negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all PPT algorithms $\mathcal{A}$ it holds that the following experiments are computationally indistinguishable*

$$\mathsf{Exp}_{\mathcal{A},\mathsf{Real}}(1^\lambda) \approx \mathsf{Exp}_{\mathcal{A},\mathsf{Ideal}}(1^\lambda).$$

Where the two experiments are defined as follows.

$\underline{\mathsf{Exp}_{\mathcal{A},\mathsf{Real}}(1^\lambda)}$*:*

- $\mathbb{A} \leftarrow \mathcal{A}(1^\lambda)$
- $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow \mathsf{Setup}(1^\lambda, \mathbb{A})$
- $(S^* \notin \mathbb{A}, m_1, \ldots, m_\ell) \leftarrow \mathcal{A}(\mathsf{pk})$
- *For all* $i \in \{1, \ldots, \ell\}$: $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}, m_i)$
- $b \leftarrow \mathcal{A}^{\mathcal{O}}(c_1, \ldots, c_\ell, \{\mathsf{sk}_i\}_{i \in S^*})$

where $\mathcal{O}$ takes as input a function $f \in \mathcal{F}$ and a set S, computes $c \leftarrow \mathsf{Eval}(\mathsf{pk}, f, (c_1, \ldots, c_\ell))$ and returns, for all $i \in S$, the shares $p_i \leftarrow \mathsf{PDec}(\mathsf{sk}_i, c, i)$.

$\underline{\mathsf{Exp}_{\mathcal{A},\mathsf{Ideal}}(1^\lambda)}$*:*

- $\mathbb{A} \leftarrow \mathcal{A}(1^\lambda)$
- $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow \mathsf{Setup}(1^\lambda, \mathbb{A})$
- $(S^* \notin \mathbb{A}, m_1, \ldots, m_\ell) \leftarrow \mathcal{A}(\mathsf{pk})$
- $(\{\mathsf{sk}_i^*\}_{i \in S^*}, \mathsf{td}) \leftarrow \mathsf{Sim}_0(\mathsf{pk})$
- *For all* $i \in \{1, \ldots, \ell\}$: $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}, m_i)$
- $b \leftarrow \mathcal{A}^{\tilde{\mathcal{O}}}(c_1, \ldots, c_\ell, \{\mathsf{sk}_i^*\}_{i \in S^*})$

where $\tilde{\mathcal{O}}$ takes as input a function $f \in \mathcal{F}$ and a set S and returns $\mathsf{Sim}_1(f, (c_1, \ldots, c_\ell), f(m_1, \ldots, m_\ell), S), \mathsf{td})$.

Construction, Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2)$ be an asymmetric bilinear group of prime order p and let $H : \{0,1\}^* \to \mathbb{G}_2$ be a hash function modeled as a random oracle. In this section we describe a THE scheme for linear predicates $f : \mathbb{Z}_p^\ell \times \mathbb{Z}_p^\ell \to \{0,1\}$, i.e., our scheme supports the evaluation of functions of the form

$$f_{\vec{y}}(\vec{x}) = \begin{cases} 1 & \text{if } \sum_{i=1}^\ell x_i y_i = 0 \\ 0 & \text{otherwise.} \end{cases}$$

The scheme is described in Figure 6.

To see why the scheme satisfies correctness, observe that

$$\begin{aligned}(c_0, c_1) &= \left(\prod_{i=1}^\ell c_{i,0}^{y_i}, \prod_{i=1}^\ell c_{i,1}^{y_i}\right) \\ &= \left(\prod_{i=1}^\ell (d_0^{m_i} \cdot g_1^{r_i})^{y_i}, \prod_{i=1}^\ell (d_1^{m_i} \cdot h^{r_i})^{y_i}\right) \\ &= \left(\prod_{i=1}^\ell (g_1^{r_i + r m_i})^{y_i}, \prod_{i=1}^\ell (g_1^{\rho m_i} \cdot h^{r_i + r m_i})^{y_i}\right) \\ &= \left(g_1^\psi, g_1^{\rho \sum_{i=1}^\ell m_i y_i} \cdot h^\psi\right)\end{aligned}$$

where $\psi = \sum_{i=1}^\ell (r_i + r m_i) y_i$. Let $k = g_2^\kappa$, for some $\kappa \in \mathbb{Z}_p$, since $(b_0, b_1) = (e(c_0, k), e(c_1, k))$ then we have that

$$\begin{aligned}\prod_{i \in S} p_i^{\lambda_i} &= \prod_{i \in S} b_0^{x_i \lambda_i} = \prod_{i \in S} (g_T^{\psi\kappa})^{x_i \lambda_i} = (g_T^{\psi\kappa})^{\sum_{i=1}^\ell x_i \lambda_i} \\ &= (g_T^{\psi\kappa})^x = h^{\psi\kappa} = b_1 = P\end{aligned}$$

for all S such that $|S| = t$, if and only if $\sum_{i=1}^\ell m_i y_i = 0$.

Analysis. In the following we show that the scheme is semantically secure.

Theorem 3 (Semantic Security)**.** *If the XDH problem is hard over $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2)$ then the THE construction as described in Figure 6 is simulation secure in the random oracle model.*

$\mathsf{Setup}(1^\lambda, (t,n))$: On input the security parameter 1^λ and a pair of threshold parameters (t,n), the setup algorithm samples a uniform triple $(x,r,\rho) \leftarrow_\$ \mathbb{Z}_p^*$ and sets the public key of the scheme to

$$\mathsf{pk} = (h, d_0, d_1) = (g_1^x, g_1^r, h^r \cdot g_1^\rho).$$

Then it samples a t-out-of-n Shamir secret sharing $(x_1, \ldots, x_n) \leftarrow \mathsf{Share}(x,t,n)$ and sets $sk_i = x_i$, for all $i \in \{1, \ldots, n\}$.

$\mathsf{Enc}(\mathsf{pk}, m)$: On input the public key $\mathsf{pk} = (h, d_0, d_1)$ and a message m, the encryption algorithm samples a uniform $s \leftarrow_\$ \mathbb{Z}_p^*$ and computes the ciphertext

$$c = (c_0, c_1) = (d_0^m \cdot g_1^s, d_1^m \cdot h^s).$$

$\mathsf{Eval}(\mathsf{pk}, f, (c_1, \ldots, c_\ell))$: On input the public key pk, an ℓ-input function $f = (y_1, \ldots, y_\ell)$, and a set of ciphertexts $(c_1, \ldots, c_\ell) = ((c_{1,0}, c_{1,1}), \ldots, (c_{\ell,0}, c_{\ell,1}))$, the evaluation algorithm returns an evaluated ciphertext

$$c = (c_0, c_1) = \left(\prod_{i=1}^{\ell} c_{i,0}^{y_i}, \prod_{i=1}^{\ell} c_{i,1}^{y_i} \right).$$

$\mathsf{PDec}(\mathsf{sk}, c, i)$: On input a secret key sk, a ciphertext $c = (c_0, c_1)$, and an index i, the partial decryption algorithm evaluates $k \leftarrow H(\mathsf{pk}, c)$, computes

$$b_0 = e(c_0, k) \text{ and } b_1 = e(c_1, k)$$

and returns the decryption share $p = (b_0^{\mathsf{sk}}, b_1, i)$.

$\mathsf{Rec}(p_1, \ldots, p_{\tilde{t}})$: On input a set of shares $(p_1, \ldots, p_{\tilde{t}}) = ((p_1, P, 1), \ldots, (p_{\tilde{t}}, P, \tilde{t}))$, the reconstruction algorithm samples an arbitrary subset of shares S of size $|S| = t$ and outputs 1 if

$$\prod_{i \in S} p_i^{\lambda_i} = P$$

and outputs 0 otherwise.

Figure 6. Our THE scheme for linear predicates.

Proof. The proof consists in the observation that a freshly computed ciphertext is of the form

$$\begin{aligned}(c_0, c_1) &= (d_0^m \cdot g_1^s, d_1^m \cdot h^s) \\ &= ((g_1^r)^m \cdot g_1^s, (h^r \cdot g_1^\rho)^m \cdot h^s) \\ &= (g_1^{r \cdot m} \cdot g_1^s, h^{r \cdot m} \cdot g_1^{\rho \cdot m} \cdot h^s) \\ &= (g_1^{r \cdot m + s}, h^{r \cdot m + s} \cdot g_1^{\rho \cdot m})\end{aligned}$$

for a uniform $s \leftarrow_\$ \mathbb{Z}_p^*$, which is a well-formed ElGamal ciphertext. Therefore semantic security follows from an invocation of the XDH assumption. □

The following shows that the scheme is simulation secure.

Theorem 4 (Simulation Security)**.** *If the XDH and the DBDH problems are hard over $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2)$ then the THE construction as described in Figure 6 is simulation secure.*

Proof. We consider without loss of generality an adversary that output a maximally corrupted set S^* of size exactly $t-1$. We gradually change the view of the adversary through a series of hybrids and then we describe the simulator $(\mathsf{Sim}_0, \mathsf{Sim}_1)$ in the end of the proof.

Hybrid 0: This is defined as in the experiment $\mathsf{Exp}_{\mathcal{A},\mathsf{Real}}(1^\lambda)$.

Hybrid 1: In this hybrid we change how the output of the honest share is computed for each query of the adversary. Let $(f = (y_1, \ldots, y_\ell), S)$ be the a query of the adversary, let (c_0, c_1) be the output of the corresponding evaluation algorithm, and let (b_0, b_1) be the pair as defined in the original PDec algorithm. For the set $S \cap S^*$ we compute the output as specified in the real experiment. For the complement of such a set, we compute

$$b^{(i)} = \left(\frac{b_1}{\prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} \cdot g_T^{\kappa\rho \cdot \sum_{i=1}^{\ell} m_i \cdot y_i}} \right)^{\lambda_i^{-1}}.$$

where $k = g_2^\kappa$, and output $(b^{(i)}, b_1, i)$. Note that this output is correctly distributed since

$$\begin{aligned}\left(b^{(i)}\right)^{\lambda_i} \cdot \prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} &= \frac{b_1}{g_T^{\kappa\rho \cdot \sum_{i=1}^{\ell} m_i \cdot y_i}} \\ &= \frac{h^{\sum_{i=1}^{\ell} r_i \cdot y_i} \cdot g_T^{\kappa\rho \cdot \sum_{i=1}^{\ell} m_i \cdot y_i}}{g_T^{\kappa\rho \cdot \sum_{i=1}^{\ell} m_i \cdot y_i}} \\ &= h^{\sum_{i=1}^{\ell} r_i \cdot y_i}\end{aligned}$$

which is exactly what we expect. It follows that this modification is purely syntactical and therefore the view of the adversary is unchanged.

Hybrid 2: In this hybrid the shares x_i, for all $i \in S^*$, are sampled uniformly from $\mathbb{Z}_p$, instead of being computed using the Share algorithm. Note that we no longer need to explicitly compute the shares of the non-corrupted parties. Since $|S^*| < t$ it follows that the view of the adversary is identical to that induced by the previous hybrid.

Hybrid 3: In this hybrid we compute all ciphertexts c_i as encryptions of zero, i.e.,

$$c_i = (g_1^{r_i}, h^{r_i})$$

where $r_i \leftarrow_\$ \mathbb{Z}_p^*$. Indistinguishability follows from a simple hybrid argument against the XDH assumption.

Hybrid 4: We define the fourth hybrid to be identical to the previous one, except that in the queries to the decryption shares of the honest parties are computed as

$$b^{(i)} = \left(\frac{b_1}{\prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} \cdot g_T^{\gamma \cdot \sum_{i=1}^{\ell} m_i \cdot y_i}} \right)^{\lambda_i^{-1}}.$$

where $\gamma \leftarrow_\$ \mathbb{Z}_p^*$ is sampled freshly for each adversarial query. Let Q be the set of queries issued by the adversary to the decryption oracle. For all $i \in \{1, \ldots, |Q|\}$ we define the following hybrid distribution

$$\left(g_2^{\kappa_1}, \ldots, g_2^{\kappa_Q}, g_T^{\rho\kappa_1}, \ldots, g_T^{\rho\kappa_{i-1}}, g_T^{\rho\kappa_i}, g_T^{\gamma_{i+1}} \ldots, g_T^{\gamma_Q}\right).$$

Note that on the one extreme we have that the distribution corresponds to the computation done in in Hybrid 3

$$\left(g_2^{\kappa_1}, \ldots, g_2^{\kappa_Q}, g_T^{\rho\kappa_1}, \ldots, g_T^{\rho\kappa_Q}\right)$$

whereas on the other extreme the distribution is identical to the computation done in Hybrid 4

$$\left(g_2^{\kappa_1}, \ldots, g_2^{\kappa_Q}, g_T^{\gamma_1}, \ldots, g_T^{\gamma_Q}\right).$$

One can easily show that that the distance between the i-th and the $(i+1)$-th hybrids is negligible by an invocation of the the DBDH assumption: Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, g_2^x, g_T^y, g_T^z)$ be the tuple taken as input by the distinguisher. The reduction sets $g_T^\rho = g_T^y$ and answers the queries $\{1, \ldots, i-1\}$ as specified in Hybrid 3 and the queries $\{i+1, \ldots, |Q|\}$ as specified in Hybrid 4, programming the random oracle to some value with know discrete logarithm κ. For the i-th query it programs the random oracle to $k = g_2^x$ and computes

$$b^{(i)} = \left(\frac{b_1}{\prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} \cdot g_T^{z \cdot \sum_{i=1}^{\ell} m_i \cdot y_i}} \right)^{\lambda_i^{-1}}.$$

Observe that if $z = xy$ then the view of the adversary is identical to that of the i-th hybrid distribution, whereas if z is uniform in $\mathbb{Z}_p^*$ then the view of the adversary is identical to that of the $(i+1)$-th hybrid. This shows that the hybrid distributions must be computationally indistinguishable.

Hybrid 5: In this hybrid we revert the change made in Hybrid 3. Indistinguishability follows from another invocation of the XDH assumption.

Hybrid 6: In this hybrid we compute the share of the honest parties using the output of the function f, i.e.,

$$b^{(i)} = \left(\frac{b_1}{\prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} \cdot g_T^{\gamma \cdot (1 - f(m_1, \ldots, m_\ell))}} \right)^{\lambda_i^{-1}}.$$

Note that if $f(m_1, \ldots, m_\ell) = 1$, then $\sum_{i=1}^{\ell} m_i \cdot y_i = 0$ and therefore

$$b^{(i)} = \left(\frac{b_1}{\prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j}} \right)^{\lambda_i^{-1}}$$

so in this case the view of the adversary is identical. On the other hand if $\sum_{i=1}^{\ell} m_i \cdot y_i \neq 0$, then

$$b^{(i)} = \left(\frac{b_1}{\prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} \cdot g_T^{\gamma}} \right)^{\lambda_i^{-1}}.$$

for some uniform $\gamma \leftarrow_\$ \mathbb{Z}_p^*$. Note that the distributions γ and $c \cdot \gamma$, where $\gamma \leftarrow_\$ \mathbb{Z}_p^*$ and $c \neq 0$, are identical. It follows that the change is purely syntactical and the view of the adversary is unchanged.

Simulator: The simulator $(\mathsf{Sim}_0, \mathsf{Sim}_1)$ is defined to be identical to the previous hybrid. This concludes our proof. □

Extensions and Generalizations. We first show that the class of linear predicates encompasses equality checks as a special case, i.e., the parties holding the shares can jointly test whether two ciphertext $\mathsf{Enc}(\mathsf{pk}, m_0)$ and $\mathsf{Enc}(\mathsf{pk}, m_1)$ encrypt the same message. This can be achieved by evaluating the linear function $f = (1, -1)$ homomorphically and then running the distributed decryption procedure. The predicate

$$f(m_0, m_1) = m_0 - m_1 = 0$$

is satisfied if and only if $m_0 = m_1$, which gives us exactly the desired functionality.

Furthermore, we show that linear predicates generalize to polynomial evaluations of any bounded degree. Let

$$\mathcal{P}(x) = c_0 + c_1 x + \cdots + c_\ell x^\ell$$

be an ℓ-degree polynomial over $\mathbb{Z}_p$. Provided with the encryptions of the coefficients $\mathsf{Enc}(\mathsf{pk}, c_0), \ldots, \mathsf{Enc}(\mathsf{pk}, c_\ell)$, the committee members can jointly check whether a point a is a root of $\mathcal{P}$ by setting $f = (1, a, \ldots, a^\ell)$ since

$$f(c_0, c_1, \ldots, c_\ell) = \sum_{i=0}^{\ell} c_i a^i = \mathcal{P}(a) = 0$$

if and only if a is indeed a root of $\mathcal{P}$. Using standard encoding techniques [28], polynomial evaluations immediately generalize to threshold predicates and constant-variables CNF/DNF formulae.

2. Missing Proofs

In the following we present the missing proofs for the Ethereum-compatible DZT.

Hardness of the Dual Diffie-Hellman Problem. We present positive evidence that that the dual Diffie-Hellman problem is hard, by establishing a bound in the generic group model [37]. We refer the reader to [8] a comprehensive introduction to the bilinear group model. Here we recall a useful lemma to prove facts about generic attackers.

Lemma 1 (Schwartz-Zippel)**.** *Let $P(X_1, \ldots, X_n)$ be a non-zero polynomial of degree $d \geq 0$ over a field $\mathbb{F}$. Then the probability that $P(x_1, ..., x_n) = 0$ for randomly chosen values $(x_1, ..., x_n)$ in $\mathbb{F}^n$ is bounded from above by $\frac{d}{|\mathbb{F}|}$.*

Theorem 5. *The dual Diffie-Hellman (dDH) problem is hard in the generic group model.*

Proof Sketch. A generic attacker is given the encoding of the base elements of either of the following distributions

$$D_0 = (g_1, g_2, g_1^x, g_1^y, g_1^z, g_1^{zy/x}, g_2^v, g_2^{xv}, g_2^w, g_2^{yw})$$

or

$$D_1 = (g_1, g_2, g_1^x, g_1^y, g_1^z, g_1^u, g_2^v, g_2^{xv}, g_2^w, g_2^{yw})$$

where all integers are uniformly chosen from $\mathbb{Z}_P^*$, and has to decide which is the case. The adversary is given oracle access to a group-operation oracle that supports linear operations in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ and pairing between elements in the source groups. We assume that each element is assigned a unique identifier, so equality can be tested locally. Note that the only way for an attacker to distinguish between the two distributions is to find a polynomial P of degree at most 2 with a non-trivial zero. More precisely, the adversary has to find some polynomial P such that

$$\begin{aligned}
&P\left(X, Y, Z, \{U, ZY/X\}, V, XV, W, YW\right) \\
&= c_0 + c_1 X + c_2 Y + c_3 Z + c_4\{U, ZY/X\} + c_5 V + c_6 XV \\
&+ c_7 W + c_8 YW + c_9 XV + c_{10} X^2 V + c_{11} XW + c_{12} XYW \\
&+ c_{13} YV + c_{14} YW + c_{15} Y^2 W + c_{16} ZV + c_{17} ZXV \\
&+ c_{18} ZW + c_{19} ZYW + c_{20}\{UV, ZYV/X\} \\
&+ c_{21}\{UXV, ZYV\} + c_{22}\{UW, ZYW/X\} \\
&+ c_{23}\left\{UYV, ZY^2V/X\right\} \\
&= 0
\end{aligned}$$

where formal variables correspond to the integers sampled by the challenger. We say that P is non-trivial if all the coefficients $(c_{20}, c_{21}, c_{22}, c_{23})$ are non-zero. By lemma 1 the probability that any non-trivial polynomial P evaluates to 0, over the random choices of the assignments, is negligibly small. □

Proof of Theorem 2. We present the proof of Theorem 2.

Proof. We assume without loss of generality that the adversary outputs a maximally corrupted subset of parties S^* of size $|S| = t - 1$. Consider the following sequence of hybrids.

Hybrid 0: This is the original experiment $\mathsf{Exp}_{\mathcal{A}}(1^\lambda)$.

Hybrid 1: In this hybrid all NIZK proofs from the honest parties are computed using the simulator. By a standard hybrid argument over the zero-knowledge property of the NIZK scheme, this hybrid is computationally indistinguishable from the previous one.

Hybrid 2: In this hybrid all NIZK proofs sent by the adversary are extracted using Ext. If any of the outputs of the extractor does not constitute a valid witness for the corresponding relation, then the experiment aborts. A standard hybrid argument shows the probability that an abort is triggered is bounded by a negligible function in the security parameter, by the simulation extractability of the NIZK scheme.

Hybrid 3: In this hybrid we change how the responses of the honest parties are computed for all queries of the adversary. Fix a query of the adversary (to any of the interfaces) and let $(\tilde{c}_0, \tilde{c}_1)$ be defined as in the Response algorithm. The output of the i-th honest party is identical to the original experiment except for

$$c_0^{(i)} = \left(\frac{\tilde{c}_1}{\prod_{j \in S^*} \left(\tilde{c}_0^{x_j}\right)^{\lambda_j} \cdot g_1^{\rho \cdot (\tilde{\alpha} - \alpha)}}\right)^{\psi \cdot \lambda_i^{-1}}$$

where $\psi \leftarrow_\$ \mathbb{Z}_p^*$ (note that $k^{(i)}$ is defined as before to be $k^{1/\psi}$) and $\tilde{\alpha}$ depends on the type query that the adversary is asking:

1) For queries to the Accept interface, $\tilde{\alpha}$ is set to α.
2) For queries to the Reject interface, $\tilde{\alpha}$ is set to $f(\alpha)$.
3) For queries to the Malicious interface, the values of $(\beta_0, \ldots, \beta_{\log(p)})$ are read from the extracted NIZK sent be the adversary, and the value of $\tilde{\alpha}$ is set to $\sum_{i=0}^{\log(p)} 2^i \cdot \beta_i$.

Observe that such a response is correctly distributed since

$$\begin{aligned}
&e\left(\left(c_0^{(i)}\right)^{\lambda_i}, k^{(i)}\right) \cdot \prod_{j \in S^*} e\left(\left(c_0^{(j)}\right)^{\lambda_j}, k^{(j)}\right) \\
&= e\left(\left(c_0^{(i)}\right)^{\lambda_i}, k^{(i)}\right) \cdot \prod_{j \in S^*} e\left(c_0^{(j)}, k^{(j)}\right)^{\lambda_j} \\
&= e\left(\left(c_0^{(i)}\right)^{\lambda_i}, k^{(i)}\right) \cdot \prod_{j \in S^*} e\left(\tilde{c}_0, k\right)^{x_j \cdot \lambda_j} \\
&= e\left(\left(\frac{\tilde{c}_1}{\prod_{j \in S^*} \left(\tilde{c}_0^{x_j}\right)^{\lambda_j} \cdot g_1^{\rho \cdot (\tilde{\alpha} - \alpha)}}\right)^{\psi}, k^{1/\psi}\right) \cdot \prod_{j \in S^*} e\left(\tilde{c}_0, k\right)^{x_j \cdot \lambda_j} \\
&= e\left(\frac{\tilde{c}_1}{g_1^{\rho \cdot (\tilde{\alpha} - \alpha)}}, k\right) = e\left(\frac{h^{\tilde{r}} \cdot g_1^{\rho \cdot (\tilde{\alpha} - \alpha)}}{g_1^{\rho \cdot (\tilde{\alpha} - \alpha)}}, k\right) = e\left(h^{\tilde{r}}, k\right)
\end{aligned}$$

for some adversarially chosen $\tilde{r}$. Here the second and the fifth equations are guaranteed to hold as otherwise they would trigger an abort of the simulator. It follows that this modification is only syntactical and the view of the adversary is identical to that of the previous hybrid.

Hybrid 4: In this hybrid the shares x_i, for all $i \in S^*$, are sampled uniformly from $\mathbb{Z}_p$, instead of being computed using the Share algorithm. Then the element h_i corresponding to the i-th honest party is computed as

$$h_i = \left(\frac{h}{\prod_{i \in S^*} h_j^{\lambda_j}}\right)^{\lambda_i^{-1}}.$$

Since $|S^*| < t$ it follows that the view of the adversary is identical to that induced by the previous hybrid.

Hybrid 5: In this hybrid we compute the ciphertexts (c_0, c_1) and (d_0, d_1) as encryptions of zero, i.e.,

$$(c_0, c_1) = (g_1^r, h^r) \text{ and } (d_0, d_1) = (g_1^s, h^s)$$

where $(r, s) \leftarrow_\$ \mathbb{Z}_p^*$. Indistinguishability follows from two invocations of the XDH assumption.

Hybrid 6: This hybrid is identical to the previous one, except that the responses of the honest parties for all adversarial queries are computed as

$$c_0^{(i)} = \left(\frac{\tilde{c}_1}{\prod_{j \in S^*} \left(\tilde{c}_0^{x_j}\right)^{\lambda_j} \cdot g_1^{\gamma \cdot (\tilde{\alpha} - \alpha)}}\right)^{\psi \cdot \lambda_i^{-1}}$$

where $\gamma \leftarrow_\$ \mathbb{Z}_p^*$ is sampled freshly for each adversarial query and $\tilde{\alpha}$ is defined as before.

Before showing a reduction against the hardness of the dDH assumption, we consider a reformulation of the problem, that is more convenient to use. I.e., we are going to construct a distinguisher for the distributions

$$D_0 = $$

$$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, g_1^x, g_1^y, g_1^{xz}, g_1^{yz}, g_2^{v/x}, g_2^v, g_2^{w/y}, g_2^w)$$

and

$$D_1 = $$

$$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, g_1^x, g_1^y, g_1^{xz}, g_1^u, g_2^{v/x}, g_2^v, g_2^{w/y}, g_2^w)$$

which are identical to the distributions defined by the dDH problem (up to renaming of the variables).

In the following we describe the reduction against the dDH problem. On input a tuple

$$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, g_1^x, g_1^y, g_1^{xz}, g_1^u, g_2^{v/x}, g_2^v, g_2^{w/y}, g_2^w)$$

sampled either from D_0 (in which case $u = yz$) or from D_1 (where u is uniformly from $\mathbb{Z}_p^*$). The reduction sets the public parameters of the scheme exactly as specified in Hybrid 5. Let Q be the set of queries issued by the adversary to the decryption oracle. The reduction samples a uniform $i \leftarrow_\$ \{1, \ldots, |Q|\}$, then answers all queries $\{i+1, \ldots, |Q|\}$ as specified in Hybrid 6, programming the random oracle to some value with know discrete logarithm κ. For queries $\{1, \ldots, i-1\}$ (and for each honest party) the reduction samples a uniform pair $(\chi_i, \mu) \leftarrow_\$ \mathbb{Z}_q^*$ and computes

$$\begin{aligned} c_0^{(i)} &= \frac{g_1^{x \cdot \tilde{r} \cdot \mathsf{sk} \cdot \chi_i \cdot \lambda_i^{-1}}}{\prod_{j \in S^*} g_1^{x \cdot \tilde{r} \cdot \mathsf{sk}_j \cdot \lambda_j \cdot \chi_i \cdot \lambda_i^{-1}} \cdot g_1^{xz \cdot \chi_i \cdot \lambda_i^{-1} \cdot (\tilde{\alpha} - \alpha)}} \\ &= \frac{\tilde{c}_1^{x \cdot \chi_i \cdot \lambda_i^{-1}}}{\prod_{j \in S^*} \left(\tilde{c}_0^{\mathsf{sk}_j}\right)^{\lambda_j \cdot x \cdot \chi_i \cdot \lambda_i^{-1}} \cdot g_1^{xz \cdot \chi_i \cdot \lambda_i^{-1} \cdot (\tilde{\alpha} - \alpha)}} \\ &= \left(\frac{\tilde{c}_1}{\prod_{j \in S^*} \left(\tilde{c}_0^{\mathsf{sk}_j}\right)^{\lambda_j} \cdot g_1^{z \cdot (\tilde{\alpha} - \alpha)}} \right)^{x \cdot \chi_i \cdot \lambda_i^{-1}} \end{aligned}$$

where $\tilde{r}$ is extracted from the corresponding NIZK of the adversary. Furthermore, the output of the random oracle is programmed to $g_2^{v \cdot \mu}$ and $k^{(i)}$ is set to $g_2^{v \cdot \mu / x \cdot \chi_i}$.

On the other hand, for the i-th query (and for each honest party) the reduction samples $\omega_i \leftarrow_\$ \mathbb{Z}_q^*$ sets

$$\begin{aligned} c_0^{(i)} &= \frac{g_1^{y \cdot \omega_i \cdot \tilde{r} \cdot \mathsf{sk} \cdot \lambda_i^{-1}}}{\prod_{j \in S^*} g_1^{y \cdot \omega_i \cdot \tilde{r} \cdot \mathsf{sk}_j \cdot \lambda_j \cdot \lambda_i^{-1}} \cdot g_1^{u \cdot \omega_i \cdot \lambda_i^{-1} \cdot (\tilde{\alpha} - \alpha)}} \\ &= \left(\frac{\tilde{c}_1}{\prod_{j \in S^*} \left(\tilde{c}_0^{\mathsf{sk}_j}\right)^{\lambda_j} \cdot g_1^{(u/y) \cdot (\tilde{\alpha} - \alpha)}} \right)^{y \cdot \omega_i \cdot \lambda_i^{-1}} \end{aligned}$$

and programs the output of the random oracle $k = g_2^w$ and sets $k^{(i)} = g_2^{w / y \cdot \omega_i}$. The reduction returns the same output of the adversary.

It is easy to see that the queries up to $i-1$ are answered according to Hybrid 5, whereas the queries after $i+1$ are answered according to Hybrid 6. Note that if $u = yz$ then the view of the adversary for the i-th query is identical to Hybrid 5, since $u/y = z$. On the other hand, if u is uniformly distributed, then the view of the adversary is identical to Hybrid 6. Thus, whenever the reduction guesses the i-th index correctly, its advantage is identical to that of the adversary. This shows that the hybrid distributions must be computationally indistinguishable.

Hybrid 7: In this hybrid we change the response of the honest parties to all adversarial queries. Specifically we compute

$$c_0^{(i)} = \left(\frac{\tilde{c}_1}{\prod_{j \in S^*} \left(\tilde{c}_0^{x_j}\right)^{\lambda_j} \cdot g_1^{\gamma \cdot \tilde{b}}} \right)^{\psi \cdot \lambda_i^{-1}}$$

where the bit $\tilde{b}$ is defined as follows:

(1) For queries to the Accept interface, we set $\tilde{b} = 0$.
(2) For queries to the Reject interface, we set $\tilde{b} = 1$.
(3) For queries to the Malicious interface, we set $\tilde{b} = 0$ if and only if $\alpha = \sum_{i=0}^{\log(p)} 2^i \cdot \beta_i$, where $(\beta_0, \ldots, \beta_{\log(p)})$ is extracted from the NIZK produced by the adversary.

First observe that if $\tilde{\alpha} = \alpha$, then the distribution is identical as $\tilde{b} = \tilde{\alpha} - \alpha = 0$. Furthermore, observe that for all constants $c \neq 0$ and $c' \neq 0$ the following distributions

$$(c, c', c \cdot \gamma) \equiv (c, c', c' \cdot \gamma)$$

where $\gamma \leftarrow_\$ \mathbb{Z}_p^*$, are identical. It follows that the view of the adversary is unchanged.

Analysis: We are now in the position of analyzing the success probability of the adversary in Hybrid 7. First observe that the public parameters of the scheme do not contain any information about α. Furthermore, the responses of the queries to the Accept and Reject interfaces are computed independently of α. On the other hand, each query to the Malicious interface reveals a single bit of information, i.e., whether $\tilde{\alpha} = \alpha$, for some adversarially chosen $\tilde{\alpha}$. Let Q be the set of queries to the Malicious interface, we can then bound the probability that the adversary guesses the correct α by $(|Q| + 1)/|D|$.

It remains to analyze the success probability that the adversary has without guessing a correct α, which is equivalent to the probability of producing a false proof for a (fresh) rejecting statement and can be bounded to a negligible value by the simulation extractability of the NIZK system. Thus, the success probability of the adversary against Hybrid 7 is bounded from above by $(|Q| + 1)/|D| + \mathsf{negl}(\lambda)$. By the above analysis, the same bound holds for an adversary playing against the original experiment, up to a negligible additive factor in the security parameter. □