A Provably-Unforgeable Threshold Schnorr Signature With an Offline Recovery Party

Michele Battagliola[1,*,†], Alessio Galli[1,†], Riccardo Longo[1,†] and Alessio Meneghetti[1,†]

[1]Department of Mathematics, University Of Trento, 38123 Povo, Trento, Italy

Abstract

The increase in the interest in cryptocurrencies, and the consequent need for technological maturity of blockchain-based platforms, has been the fuel for some recent advances in cryptographic research. In this context, digital signature protocols have a central role since they guarantee ownership and control of digital assets.

The absence of trusted central authorities in public blockchains, which is the very foundation of this technology, poses some interesting challenges on the management of digital identities. In particular, the computational infeasibility of restoring a lost key is a threat to anyone possessing this kind of digital assets. A possible solution to this problem is to use threshold multi-signatures, partially relying on a recovery-party whose only role, even though of paramount importance, is to intervene in case of key loss.

We present a Schnorr multi-party digital signature scheme that supports an offline participant during the key-generation phase, without relying on a trusted third party. Under standard assumptions we prove our scheme secure against adaptive malicious adversaries and capable of achieving the resiliency of the recovery in the presence of a malicious party.

Keywords

94A60 cryptography, 12E20 finite fields, 14H52 elliptic curves, 94A62 authentication and secret sharing, 68W40 analysis of algorithms.

1. Introduction

Custody of cryptocurriencies, and in general of crypto-assets, is at the very core of the burgeoning digital-asset market. Ownership is guaranteed by digital signatures and making them available and usable by the general public presents many issues: to provide a few examples, in case of inheritance heirs cannot access the crypto-assets unless unless they already have access to the private key, and, in general, private keys can be easily lost or forgotten, leading to the inaccessibility of the related assets. Many solutions have been devised to mitigate these problems and to enable safe custody. Some rely on

DLT 2022: 4th Distributed Ledger Technology Workshop, June 20, 2022, Rome, Italy

*Corresponding author.

†These authors contributed equally.

battagliola.michele@gmail.com (M. Battagliola); alessio.galli014@gmail.com (A. Galli); riccardolongomath@gmail.com (R. Longo); alessio.meneghetti@unitn.it (A. Meneghetti)

0000-0002-8269-2148 (M. Battagliola); 0000-0003-2104-7871 (A. Galli); 0000-0002-8739-3091 (R. Longo); 0000-0002-5159-7252 (A. Meneghetti)

personal efforts (e.g., cold storage), others simply delegate full control of the assets to a third party. Unfortunately, these kinds of solutions are partial: most either sacrifice usability or completely rely on the trustworthiness of a third party. An alternative and viable solution is to use threshold digital signatures [8]. This kind of technique addresses more comprehensively the problems above. It relies on multiple private keys, instead of a single one, which are distributed among parties, and a subset of them are required to control the crypto-assets. This approach is resilient with respect to the unavailability or loss of one party. In particular we design a three-parties protocol, that allows users to distribute their key to a custodian and a third party, like a bank or another financial institute. Security is guaranteed as long as the two helping parties do not collude, i.e., it is sufficient that one of the two remains honest to preserve the safety of the system. Furthermore, this solution is effectively agnostic to the underlying blockchain, i.e., it does not have to be supported by special features.

Starting from the highly influential work of Gennaro et al [14], several authors proposed both novel schemes [7, 19, 20] and improvements to existing protocols [4, 9, 10, 12, 13, 17, 18, 21].

Recently, in [1] and [2] the authors propose an ECDSA-compatible and an EdDSA-compatible $(2, 3)$-threshold multi-signature protocol in which one of the users plays the role of the recovery party: a user involved only once in a preliminary setup prior even to the key-generation step of the protocol.

In this paper we propose a third, Schnorr-based, variant of [2]. The Schnorr signature algorithm has recently gained popularity in the world of cryptocurrencies, especially since its addition to Bitcoin with BIP340[1]. Schnorr signatures have many advantages, such as linearity, non-malleability and provable security. In particular, they are strongly unforgeable under chosen message attacks: in the random oracle model assuming the hardness of the discrete logarithm problem, in the generic group model assuming variants of preimage and second preimage resistance of the used hash function. In contrast, the best known results for the provable security of ECDSA rely on stronger assumptions. Moreover, the threshold version presented here allows for fast computation with fewer rounds of communication with respect to ECDSA, and unlike EdDSA does not require expensive computation to derive a deterministic nonce.

We prove the protocol secure against adaptive adversaries by reducing it to the classical Schnorr scheme, assuming the security of a non-malleable commitment scheme, and an IND-CPA encryption scheme. Moreover we make some considerations about the resiliency of the recovery, an interesting aspect due to the presence of an offline party, analyzing possible changes that allow us to achieve this higher level of security.

2. Preliminaries

In this section we present some preliminary definitions and primitives that will be used in the protocol and its proof of security.

[1] see https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki

Notation We use the symbol $\|$ to indicate the concatenation of bit-strings. Sometimes we slightly abuse the notation and concatenate a bit-string M with a group element $\mathscr{P}$, in those cases we assume that there has been fixed an encoding φ that maps group elements into bit-strings, so $M\|\mathscr{P} := M\|\varphi(\mathscr{P})$.

In the following when we say that an algorithm is efficient we mean that it runs in (expected) polynomial time in the size of the input, possibly using a random source.

We use a blackboard-bold font to indicate algebraic structure (i.e. sets, groups, rings, fields), a calligraphic font will generally denote elements of a finite group.

2.1. Cryptographic Hash Functions

In the Schnorr scheme (and therefore in our threshold protocol) a cryptographic hash function H is used as a Pseudo-Random Number Generator (PRNG), employed to derive secret scalars and nonces. The requirements needed for the hash function used in Schnorr signatures are analyzed in [23].

2.1.1. Schnorr Signature

Schnorr's digital signature algorithm is an efficient algorithm able to generate short signatures without sacrificing security. It is one of the first signatures that bases its security on the difficulty of discrete logarithm problem [24].

If Alice wants to send a signed message to Bob, she has to choose group $\mathbb{G}$ with generator g of prime order q where the discrete logarithm problem is considered to be hard and a cryptographic hash function H. Then they can do the following:

1. Key Generation: Alice chooses randomly a private key $x \in \mathbb{Z}_q^*$ and computes the public key $y = g^x$;
2. Signature Generation: to sign a message m, Alice performs the following:
   a) Choose randomly $k \in Z_q^*$;
   b) Compute $r = g^k$;
   c) Compute $e = H(m\|r)$;
   d) Compute $s = (k - xe)$;
   e) The signature is the pair (e, s).
3. Signature Verification: to verify the signature after receiving m and (e, s), Bob performs the following:
   a) Compute $r_v = g^s y^e$;
   b) Compute $e_v = H(m\|r_v)$;
   c) The signature is valid only if $e_v = e$.

2.2. Encryption Scheme

In our protocol we need an asymmetric encryption scheme to communicate with the offline party. The minimum requirement we ask for our protocol to be secure is that the encryption scheme chosen by the offline party has the property of IND-CPA [3, 22].

This hypothesis will be enough to prove the unforgeability of the protocol, but it is possible to achieve a higher notion of security by using a more sophisticated encryption scheme that supports Zero-Knowledge Proofs for the Discrete Logarithm. This will be more clearly explained in Section 4.5.

2.3. Commitment Schemes

A commitment scheme [5] is composed by two algorithms:

- $\mathsf{Com}(M) : \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$: takes in input the value M to commit[2] and, using a random source, outputs the commitment string C and the decommitment string D.
- $\mathsf{Ver}(C, D) : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$: takes the commitment and decommitment strings C, D and outputs the originally committed value M if the input pair is valid, $\perp$ otherwise[3].

We require a commitment scheme to have the following properties:

- Correctness: for every value M it holds $\mathsf{Ver}(\mathsf{Com}(M)) = M$.
- Binding: for every commitment string C it is infeasible to find $M \neq M'$ and $D \neq D'$ such that $\mathsf{Ver}(C, D) = M$ and $\mathsf{Ver}(C, D') = M'$ with both $M, M' \neq \perp$.
- Hiding: Let $(C, D) = \mathsf{Com}(M_b)$ with $b \in \{0,1\}$, $M_1 \neq M_0$, then it is infeasible for an attacker that may choose $M_0 \neq M_1$ and sees only C, to correctly guess b with more than negligible advantage.
- Non Malleability: Given $C = \mathsf{Com}(M)$, it is infeasible for an adversary to produce another commitment string C' such that after seeing D such that $\mathsf{Ver}(C, D) = M$, can find a decommit string D' such that $\mathsf{Ver}(C', D') = M'$ with M' related to M, that is can only create commitments to values that are independent from M.

2.4. Zero-Knowledge Proofs

In the protocol, various Zero-Knowledge Proofs (ZKP) [16] are used to enforce the respect of the passages prescribed by the specifications. In fact, in the proof of security we can exploit the soundness of these sub-protocols to extract valuable information from the

[2] In the protocol and the simulations we implicitly encode every value we need to commit into a bit-string, assuming there is a standard encoding understood by all parties

[3] Again, in the protocol we implicitly decode valid decommitment outputs (i.e. $\neq \perp$) into the original value, assuming that the decoding is also standard and understood by all parties

adversary, and their zero-knowledge property to simulate correct executions even without knowing some secrets. We can do so because we see the adversary as a (black-box) algorithm that we can call on arbitrary input, and crucially we have the faculty of rewinding its execution.

In particular we use ZKP of Knowledge (ZKPoK) to guarantee the usage of secret values that properly correspond to the public counterpart, specifically the Schnorr protocol for discrete logarithms, and its variant that proves that two public values are linked to the same secret (see [24, 27]). The soundness property of a ZKPoK guarantees that the adversary must know the secret input, and appropriate rewinds and manipulations of the adversary's execution during the proof allows us to extract those secrets and use them in the simulation. Conversely exploiting the zero-knowledge property we can trick the adversary in believing that we know our secrets even if we do not, thus we still obtain a correct simulation of our protocol form the adversary's point of view.

2.5. Feldman-VSS

Feldman's VSS scheme [11] is a verifiable secret sharing scheme built on top of Shamir's scheme [26]. A secret sharing scheme is verifiable if auxiliary information is included, that allows players to verify the consistency of their shares. We use a simplified version of Feldman's protocol: if the verification fails the protocol does not attempt to recover excluding malicious participants, instead it aborts altogether. In a sense we consider somewhat honest participants, for this reason we do not need stronger schemes such as [15, 25].
The scheme works as follows:

1. A cyclic group $\mathbb{G}$ of prime order q is chosen, as well as a generator $g \in \mathbb{G}$. The group $\mathbb{G}$ must be chosen such that the discrete logarithm is hard to compute.

2. The dealer computes a random polynomial P of degree t with coefficients in $\mathbb{Z}_q$, such that $P(0) = s$ where $s \in \mathbb{Z}_q$ is the secret to be shared.

3. Each of the n share holders receive a value $P(i) \in \mathbb{Z}_q$. So far, this is exactly Shamir's scheme.

4. To make these shares verifiable, the dealer distributes commitments to the coefficients of P. Let $P(X) = s + \sum_{i=1}^{n} a_i X^i$, then the commitments are $\mathscr{C}_0 = g^s$ and $\mathscr{C}_i = g^{a_i}$ for $i \in \{1, \dots, n\}$.

5. Any party can verify its share in the following way: let α be the share received by the i-th party, then it can check if $\alpha = P(i)$ by verifying if the following equality holds:
$$g^{\alpha} = \prod_{j=0}^{t} (\mathscr{C}_j)^{i^j} = g^s \cdot g^{\sum_{j=1}^{t} a_j (i^j)} = g^{s + \sum_{j=1}^{t} a_j (i^j)} = g^{P(i)}.$$

3. Threshold Schnorr Signature

In this section we describe the main protocol: a $(2,3)$-threshold variant of Schnorr digital signature algorithm with an offline participant. Let P_1, P_2, P_3 the parties involved in the protocol, as already mentioned the goal is to allow to one of them, namely P_3 to remain offline during the key generation phase. Moreover our goal is to allow for a trustless setup, where the parties does not have to rely to a third trusted party to generate the credential. From now on we refer to P_3 as the offline or recovery party, since its role is to take part in the signing protocol if for any reason one of the two is no more able (secret key loss, unreachability, etc.).

The protocol is dividend into four algorithms:

1. Setup Phase (3.1): in this phase all three players interact to set some common parameters. Note that in a practical implementation this phase can be performed ahead of time without any real communication, because these parameters are usually fixed (e.g. for Bitcoin applications which have to use secp256k1 and SHA-256).

2. Key-Generation (3.2): performed by P_1 and P_2 to create the public key for the signature scheme and the private shards for themselves and P_3;

3. Ordinary Signature (3.3): used whenever P_1 and P_2 want to perform a signature. It is called ordinary signature as this should be the standard signing procedure;

4. Recovery Signature (3.4): ideally, this algorithm is executed when either P_1 or P_2 is no more able to sign. P_3 steps in and performs a signature with the remaining party. It is important to emphasize that the final signature is still a standard one, same as the one generated in an ordinary signature and indistinguishable to one obtained in the centralized protocol.

From now on "P_i does something" means that both P_1 and P_2 perform that action. Also by saying "P_i sends a message to P_j" means that P_1 sends data to P_2 and viceversa.

3.1. Setup Phase

The aim of this phase is to make P_1 and P_2 agree on all the parameters required in the protocol and set up the private/public key pair used to contact P_3 in case of need.

Player 1 and 2	
Input:	
Private Output:	
Public Output:	$\mathbb{G}, g, q, H$

Player 3	
Input:	
Private Output:	sk_3
Public Output:	pk_3

P_3 picks a key pair $(\mathsf{pk}_3, \mathsf{sk}_3)$ for a suitable asymmetric encryption algorithm. Then P_1 and P_2 agree on a secure hash function H whose outputs can be interpreted as elements of $\mathbb{Z}_q$ and a group $\mathbb{G}$ with generator g of prime order q in which the discrete logarithm problem is considered to be hard.

3.2. Key generation

This part of the protocol is performed by P_1 and P_2 to produce a common public key $\mathscr{A}$ and to distribute shards of the corresponding private key to each player.

Player 1	
Input:	pk_3
Private Output:	ω_1
Public Output:	$\mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}, \mathscr{A}$

Player 2	
Input:	pk_3
Private Output:	ω_2
Public Output:	$\mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}, \mathscr{A}$

1. Secret key generation and communication:
   a) P_i randomly chooses $a_i, y_{3,i}, m_i \in \mathbb{Z}_q$ and sets $\mathscr{A}_i = g^{a_i}, \mathscr{Y}_{3,i} = g^{y_{3,i}}$;
   b) P_i computes $[\mathtt{KGC}_i, \mathtt{KGD}_i] = \mathrm{Com}((\mathscr{A}_i, \mathscr{Y}_{3,i}))$;
   c) P_i sends $\mathtt{KGC}_i$ to P_j;
   d) P_i sends $\mathtt{KGD}_i$ to P_j;
   e) P_i gets $((\mathscr{A}_i, \mathscr{Y}_{3,i})) = \mathrm{Ver}(\mathtt{KGC}_j, \mathtt{KGD}_j)$.
2. Feldman VSS and generation of P_3 data:
   a) P_i sets $f_i(x) = a_i + m_i x$ and computes $y_{i,1}, y_{i,2}, y_{i,3}$ where $y_{i,j} = f_i(j)$;
   b) P_i encrypts $y_{i,3}, y_{3,i}$ with pk_3 and obtains $\mathsf{rec}_{i,3}$;
   c) P_i sends $y_{i,j}$ and $\mathsf{rec}_{i,3}$ to P_j;
   d) If the asymmetric encryption algorithm supports DLOG verification, the encryption $\mathsf{rec}_{i,3}$ is accompanied by two NIZKPs: the first one proves that the first ciphertext in $\mathsf{rec}_{i,3}$ is the encryption of the DLOG of $\mathscr{Y}_{i,3} = \mathscr{A}_i \cdot (\mathscr{M}_i)^3$ (where $\mathscr{M}_i = g^{m_i}$ is sent during the Feldman-VSS protocol), the second NIZKP proves that the second ciphertext is the encryption of the DLOG of $\mathscr{Y}_{3,i}$. P_i checks the NIZKPs attached to $\mathsf{rec}_{j,3}$.
   e) P_i checks, as in the Feldman-VSS protocol, the integrity and consistency of the shards $y_{j,i}$;
   f) P_i computes $x_i = y_{1,i} + y_{2,i} + y_{3,i}$.
3. P_i proves in ZK the knowledge of x_i using Schnorrs protocol.
4. Public key and shards generation:
   a) the public key is $\mathscr{A} = \prod_{i=1}^{3} \mathscr{A}_i$, where $\mathscr{A}_3 = (\mathscr{Y}_{3,1})^2 / \mathscr{Y}_{3,2}$ so that $a_3 = 2y_{3,1} - y_{3,2}$. From now on we will set $a = \sum_{i=1}^{3} a_i$ and we have $g^a = \mathscr{A}$;
   b) P_1 computes $\omega_1 = 2x_1$, while P_2 computes $\omega_2 = -x_2$. Note that $\omega_1 + \omega_2 = a$;

3.3. Signature Algorithm

This algorithm is the general signature scheme in which two players, P_A and P_B, want to sign a message. Each of P_1, P_2, P_3 can take the role of either P_A or P_B depending on the situation, we call Ordinary Signature the case in which P_1 takes the role of P_A and P_2 takes the role of P_B.

Let M be the message, the parameters involved are:

Player A	
Input:	$M, \omega_A, \mathscr{A}$
Public Output:	(s, e)

Player B	
Input:	$M, \omega_B, \mathscr{A}$
Public Output:	(s, e)

The protocol proceeds as follows.

1. Generation of r:
   a) P_i randomly chooses $k_i \in \mathbb{G}$;
   b) P_i computes $r_i = g^{k_i}$;
   c) P_i computes $[\mathtt{KGC}_i, \mathtt{KGD}_i] = \mathrm{Com}(r_i)$ and sends $\mathtt{KGC}_i$;
   d) P_i sends $\mathtt{KGD}_i$;
   e) P_i computes $r_j = \mathrm{Ver}([\mathtt{KGC}_j, \mathtt{KGD}_j])$;
   f) P_i computes $r = r_A r_B$.
2. Generation of s:
   a) P_i computes $e = H(r \| M)$ and $s_i = k_i - \omega_i e$;
   b) P_i computes $[\mathtt{KGC}'_i, \mathtt{KGD}'_i] = \mathrm{Com}(s_i)$ and sends $\mathtt{KGC}'_i$;
   c) P_i sends $\mathtt{KGD}'_i$;
   d) P_i computes $s_j = \mathrm{Ver}([\mathtt{KGC}'_j, \mathtt{KGD}'_j])$;
   e) P_i computes $s = s_A + s_B$.
3. P_i computes $r_v = g^s \mathscr{A}^e$ and checks that $H(r_v \| M) = e$.

The output signature is (s, e). If a check fails, the protocol aborts.

3.4. Recovery Signature

This is the scenario where one between P_1 or P_2 is unable to sign. P_3 has to come back online and perform a recovery signature with the other online party. There are two different situations, depending whether the other party is P_1 or P_2.

Firstly we consider the case where P_2 is offline and P_1 and P_3 want to perform a signature. The parameters involved are:

Player 1	
Input:	$M, \omega_1, \mathscr{A}, \mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}$
Public Output:	(s, e)

Player 3	
Input:	M, sk_3
Public Output:	(s, e)

The protocol is the following.

1. Communication:
   a) P_1 contacts P_3 and sends $\mathscr{A}, \mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}$;
   b) P_3 decrypts $\mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}$ using its private key to obtain $y_{1,3}, y_{3,1}, y_{2,3}, y_{3,2}$;
   c) P_3 computes $a_3 = 2y_{3,1} - y_{3,2}$ and $\mathscr{A}_3 = g^{a_3}$.
2. P_3's key creation:
   a) P_3 computes $x_3 = y_{1,3} + y_{2,3} + 2y_{3,2} - y_{3,1}$;
   b) P_i proves in ZK the knowledge of x_i using Schnorrs protocol ($x_1 = \frac{1}{2}\omega_1$).
3. Signature generation:
   a) P_1 computes $\tilde{\omega}_1 = \frac{3}{4}\omega_1$;
   b) P_3 computes $\omega_3 = -\frac{1}{2}x_3$;
   c) P_1 and P_3 perform the signature algorithm with $P_A = P_1$, $P_B = P_3$ using $\omega_A = \tilde{\omega}_1$ and $\omega_B = \omega_3$. Note that it still holds that $\omega_A + \omega_B = a$.

The other scenario is the one in which P_1 is offline and P_2 signs the message with P_3:

Player 2	
Input:	$M, \omega_2, \mathscr{A}, \mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}$
Public Output:	(s, e)

Player 3	
Input:	M, sk_3
Public Output:	(s, e)

The first two steps are the same as in the previous scenario, except that in the ZKP in [2b] we now have $x_2 = -\omega_2$.

3. Signature generation:
   a) P_2 computes $\tilde{\omega}_2 = -3\omega_2$;
   b) P_3 computes $\omega_3 = -2x_3$;
   c) P_2 and P_3 perform the signature algorithm with $P_A = P_2$, $P_B = P_3$ using $\omega_A = \tilde{\omega}_2$ and $\omega_B = \omega_3$. Note that also here $\omega_A + \omega_B = a$.

4. Security Proof

In this section we discuss the security of the scheme in terms of the unforgeability properties defined below. We also discuss other security aspects, such as recovery resiliency in the subsequent Section 4.5.

Definition 4.1 (Unforgeability). A (t, n)-threshold signature scheme is unforgeable if no malicious adversary who corrupts at most $t - 1$ players can produce the signature on a new message m with non-negligible probability, given the view of the threshold sign on input messages $m_1, \ldots, m_k$ (adaptively chosen by the adversary), as well as the signatures on those messages.

The unforgeability of our protocol is formally stated in the following theorem:

Theorem 4.1. Assuming that:

- the Schnorr signature scheme instantiated on the group $\mathbb{G}$ of prime order q with the hash function H is unforgeable;
- $\mathsf{Com}, \mathsf{Ver}$ is a non-malleable commitment scheme;
- the Decisional Diffie-Hellman Assumption holds;
- the encryption algorithm used by P_3 is IND-CPA;

our threshold protocol is unforgeable.

In Section 4.4 we will prove the theorem by showing that if there is an adversary able to forge a signature for the threshold scheme with non negligible probability $\epsilon > \lambda^{-c}$ with λ a polynomial and $c > 0$, then it is possible to build a forger $\mathcal{F}$ that forges a signature for the centralized Schnorr scheme also with non negligible probability. The simulation works by having an oracle that feeds inputs for the centralized scheme to $\mathcal{F}$, our goal is to respond by generating a signature exploiting . First, it has to simulate the key generation protocol in order to match the key received from the oracle, then it can proceed with the signature part. The core of this setup is that if is able to crack our protocol, $\mathcal{F}$ will take advantage of that and will also create a forgery for the centralized version of the oracle.

Following the definition of unforgeability, will control one player while $\mathcal{F}$ controls the remaining two. We must consider two different scenarios: one where controls P_1 or P_2, and the case where controls P_3. First, we suppose without loss of generality that controls P_2.

The adversary interacts by first participating in the key generation part to generate a public key $\mathcal{A}$, then starts requesting signatures on some messages $m_1, \dots, m_l$. Here it can either take part in the process or let P_1 and P_3 generate the signature. Eventually outputs a message $m \neq m_i \ \forall i$ and its valid signature with probability at least ϵ, where this is taken over the random tapes of the adversary and the honest player, respectively τ and τ_i. So we can write that

$$\mathbb{P}_{\tau_i, \tau_{\mathcal{A}}}(\quad (\tau \ )_{P_i(\tau_i)} = \text{forgery}) \geq \epsilon, \tag{1}$$

where $(\tau \ )_{P_i(\tau_i)}$ is the output of at the end of this process and $\mathbb{P}_{\tau_i, \tau_{\mathcal{A}}}$ denotes that the probability is taken over the random tape τ_i and the adversary tape τ .

We say that a random tape is good if

$$\mathbb{P}_{\tau_i}(\quad (\tau \ )_{P_i(\tau_i)} = \text{forgery}) \geq \frac{\epsilon}{2}. \tag{2}$$

We recall the following useful lemma, stated and proved in [2].

Lemma 4.1. If τ is chosen uniformly at random, the probability that τ is good is at least $\frac{\epsilon}{2}$.

4.1. Key generation simulation

Now we see into details how the key generation phase is simulated. $\mathcal{F}$ receives from the challenger the public key $\mathscr{A}_c$ for the centralized Schnorr protocol and a public key pk_3 for the asymmetric encryption scheme. The simulation proceeds as follows:

1. P_i selects random values $a_i, y_{3,i}, m_i \in \mathbb{Z}_q$ and computes $\mathscr{A}_i = g^{a_i}, \mathscr{Y}_{3,i} = g^{y_{3,i}}$;
2. P_i computes the commitment $[\mathtt{KGC}_i, \mathtt{KGD}_i] = \mathsf{Com}(\mathscr{A}_i, \mathscr{Y}_{3,i})$;
3. P_i sends $\mathtt{KGC}_i$, then, after receiving $\mathtt{KGC}_j$, P_i sends $\mathtt{KGD}_i$;
4. P_i gets $(\mathscr{A}_i, \mathscr{Y}_{3,i}) = \mathsf{Ver}(\mathtt{KGC}_j, \mathtt{KGD}_j)$;
5. Now $\mathcal{F}$ knows all the parameters needed in the computation of $\mathscr{A}$, so it rewinds to step 3, aiming to get $\mathscr{A} = \mathscr{A}_c$;
6. $\mathcal{F}$ computes $\hat{\mathscr{A}} = \frac{\mathscr{A}_c}{\mathscr{A}_2 \mathscr{A}_3}$, computes the commitment $[\hat{\mathtt{KGC}}_1, \hat{\mathtt{KGD}}_1] = \mathsf{Com}(\hat{\mathscr{A}}, \mathscr{Y}_{3,1})$, and sends it to , so that it will receive $\hat{\mathscr{A}}$ as $\mathscr{A}_1$ which leads to $\mathscr{A} = \mathscr{A}_c$;
7. $\mathcal{F}$ simulates a fake Feldman-VSS with $\mathscr{A}$ (see e.g. [2]) since it cannot compute the polynomial $f(x)$: it selects $y_{1,2}, y_{1,3}$ randomly and computes $c_{1,j} = \frac{1}{i}\left(g^{y_{1,i}} / \hat{\mathscr{A}}\right)$.
8. P_i encrypts $y_{i,3}$ and $y_{3,i}$ with pk_3, getting $\mathsf{rec}_{i,3}$, then sends $y_{i,j}, \mathsf{rec}_{i,3}$;
9. P_i computes x_i. Since $\mathcal{F}$ does not know the discrete logarithm of $\hat{\mathscr{A}}$, it sets x_1 randomly;
10. $\mathcal{F}$ participates in the ZK proofs rewinding and selecting appropriate challenges in order to extract x_2 from ;
11. P_j can compute the key $\mathscr{A}$ as described in the enrollment phase. can also compute ω_2, while $\mathcal{F}$ cannot, since it does not know x_1.

Note that at the end of the protocol, $\mathcal{F}$ does not know x_1 nor ω_1, but $\mathcal{F}$ will still be able to complete correctly the signing part by querying the oracle.

The proof of the correctness of the simulation is stated in the following lemmas. The proofs are trivial and use the same argument of the one presented in [2].

Lemma 4.2. If the Decisional Diffie-Hellman assumption holds, and the encryption algorithm used by P_3 is IND-CPA, then the simulation terminates in expected polynomial time and is indistinguishable from the real protocol.

Lemma 4.3. For a polynomial number of inputs the simulation terminates with output $\mathscr{A}_c$ except with negligible probability.

Observation 1. It is important that in step 3 the adversary sends $\texttt{KGC}_2$ and $\texttt{KGD}_2$ before $\mathcal{F}$, so that after the rewinding cannot change its commitment (note that this applies also to the simulation in Section 4.2). If the order were inverted, could also use the commitment of $\mathcal{F}$ to generate its value. Assuming the non-malleability property, does not deduce anything about the content of the commitment, but it could still use it as a seed for a random generator. If this were to happen, $\mathcal{F}$ can guess $\hat{\mathcal{A}}$ with probability $\frac{1}{q}$ with q the size of the group, making the expected time exponential.

It is possible to swap the order in the commitment step using an equivocable commitment scheme with a secret trapdoor. In this case we only need to rewind at the decommitment step and change KCD_1 in order to match $\hat{\mathcal{A}}$.

4.2. Signature generation simulation

After the the key generation, $\mathcal{F}$ has to deal with the signature requests issued by . When asks for a signature, $\mathcal{F}$ performs a simulation while having access to the signing oracle that uses the previously created public key. Here $\mathcal{F}$ can fully predict what will output and, while it does not know any secret key of P_1, it knows everything of P_2 since all the secret values were extracted from during the ZK proofs.

The simulation proceeds as follows:

1. chooses a message m to sign;
2. $\mathcal{F}$ queries its signing oracle for a signature for m corresponding to the public key $\mathcal{A}$ and gets (s_f, e_f);
3. P_i randomly chooses $k_i \in \mathbb{Z}_q^*$, then computes $r_i = g^{k_i}$ and $[\texttt{KGC}_i, \texttt{KGD}_i] = \text{Com}(r_i)$;
4. P_i sends $\texttt{KGC}_i$, then, after receiving $\texttt{KGC}_j$, sends $\texttt{KGD}_i$ and gets $r_i = \text{Ver}([\texttt{KGC}_i, \texttt{KGD}_i])$;
5. $\mathcal{F}$ rewinds to step 4;
6. $\mathcal{F}$ computes $\hat{r}_1 = \frac{r_f}{r_2}$, then its commitment $[\hat{\texttt{KGC}}_1, \hat{\texttt{KGD}}_1] = \text{Com}(\hat{r}_1)$ and sends $\hat{\texttt{KGC}}_1$ to so it receives $\hat{r}_1$ as r_1 which leads to $r = r_f$;
7. P_i computes $r = r_1 r_2$, $e = H(r \| m)$, and $s_i = k_i - \omega_i e$ ($\mathcal{F}$ picks s_1 at random);
8. P_i computes $[\texttt{KGC}'_i, \texttt{KGD}'_i] = \text{Com}(s_i)$, then sends $\texttt{KGC}'_i$;
9. P_i sends $\texttt{KGD}'_i$ and gets $s_i = \text{Ver}([\texttt{KGC}'_i, \texttt{KGD}'_i])$;
10. $\mathcal{F}$ computes $r'_2 = g^{s_2} \cdot g^{-e\omega_2}$, if $r_2 = r'_2$ it rewinds to step 8, otherwise it sends s_1 and aborts;
11. $\mathcal{F}$ computes $\hat{s}_1 = s_f - s_2$ with its commitment $[\hat{\texttt{KGC}}'_1, \hat{\texttt{KGD}}'_1] = \text{Com}(\hat{s}_1)$ and sends $\hat{\texttt{KGC}}'_1$ to so it receives $\hat{s}_1$ as s_1 which leads to $s = s_f$;
12. P_i computes $s = s_1 + s_2$ and $r_v = g^s \mathcal{A}^e$, then checks that $H(r_v \| m) = e$. If a check fails the protocol aborts, otherwise the signature is (s, e).

Lemma 4.4. If Com is a secure non-malleable commitment scheme, the protocol above is a perfect simulation of the centralized one and terminates correctly with output (s_f, e_f).

Proof. The simulation is identical to the real protocol except that here $\mathcal{F}$ does not know its secret shards. Nevertheless it is still able to retrieve the correct value from by rewinding it. As above, if the protocol terminates, by construction it will terminate with output (s_f, e_f). If is dishonest or refuses to decommit some values, the protocol aborts. Note that the check of step 10 is introduced to preserve any abort that the adversary may cause by sending an invalid s_1. □

4.3. Recovery signature simulation

Since can ask both types of signature, we must also consider the case of a recovery signature. The core algorithm remains the same, so the results above still holds. Here we only need to change the setup phase during which the third player recovers its secret data. There are two scenarios: one in which controls one of P_1 or P_2 and another where it controls P_3, which is easier, since the enrollment phase can be avoided. We will proceed in order.

Trivially, if asks for a recovery signature between the two honest parties, $\mathcal{F}$ can simply ask its oracle and output whatever it received from the oracle. So we can limit ourselves to deal with the case where participates in the signing process.

If controls P_2 the simulation works as follows:

1. P_2 sends to P_3 $\mathcal{A}, \mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}$. Note that some of them are random data sent by P_1;
2. P_3 cannot decrypt the values received in the previous step. It simulates the ZKP about x_3 and extracts the secret values from P_2;
3. P_2 computes $\tilde{\omega_2} = -3\omega_2$. Note that P_3 does not have the right shards so it cannot compute its secret key;
4. They perform the signing algorithm using the simulation above. Here $\mathcal{F}$ does not know its secret key, but it can use the signing oracle to get the signature.

If controls P_1 the only difference is in the computation of $\tilde{\omega}_1 = \frac{3}{4}\omega_1$. The last case is the one where controls P_3. The enrollment phase is done all by $\mathcal{F}$ so it can easily generate random shards that will be sent to P_3 during the recovery signature phase and output the public key given by the oracle. Then with the same simulation as before it can simulate the signature.

4.4. Proof of the unforgeability property

Now that we have dealt with all the possible cases we need to prove Theorem 4.1:

Proof. Let $Q < \lambda^c$ be the maximum number of signature queries that the adversary makes. In a real instance of the protocol the adversary outputs a forgery after $l < Q$ queries, either because it stops submitting queries or because the protocol aborts. As we previously proved, our simulator produces a view of the protocol that the adversary cannot distinguish from the real one, therefore will produce a forgery with the same probability as in a real execution. Then the probability of success of our forger $\mathcal{F}$ is $\frac{\epsilon^3}{8}$ which is the product of the probability of the following independent events:

1. choosing a good random tape for , whose probability is at least $\frac{\epsilon}{2}$ as per Lemma 4.1;
2. getting a good public key, whose probability is at least $\frac{\epsilon}{2}$ as shown in Lemma 4.2 and 4.3;
3. successfully produces a forgery, whose probability is again $\frac{\epsilon}{2}$ (2).

Under the assumption on the security of the Schnorr signature scheme, the probability of success of $\mathcal{F}$ must be negligible, which implies that ϵ must be negligible too, contradicting the hypothesis that has a non-negligible probability of forging a signature for the scheme. □

4.5. Resilience of the recovery

In our security analysis we focused on the unforgeability of the signature, however with an offline party another security aspect is worthy of consideration: the resiliency of recovery in the presence of a malicious adversary. Of course if the offline party is malicious and unwilling to cooperate there is nothing we can do about it, however the security can be strengthened if we consider that one of the online parties may corrupt the recovery material. In this case a generic CPA asymmetric encryption scheme is not sufficient to prevent malicious behaviour, because we need a verifiable encryption scheme that allows the parties to prove that the recovery material is consistent, just like they prove that they computed the shards correctly.

In particular we need an encryption scheme that supports DLOG verification as explained in point 2d of the Key-Generation algorithm. A suitable candidate could be a variant of the CramerShoup cryptosystem presented in [6]. This algorithm is equipped with a ZKP that allows the sender to prove that the plaintext encrypted is the discrete logarithm of a public value. In particular, since the protocol is a three step ZKP with special soundness, completeness, and honest-verifier zero knowledge, it is possible to build a non-interactive ZKP using the Fiat-Shamir heuristic.

5. Conclusions

In this paper, we presented a Schnorr threshold signature with the goal of providing a reliable and efficient solution for the custody of crypto-assets, both from possible attackers and from loss due to accidents of various nature. In this sense, threshold signatures

without a trusted dealer offer a perfect solution, since the private key is never created, and they overcome the limitations of blockchains that do not have native multi-signature support. Although decentralized signature algorithms have been known for a while, we are aware of only few proposals for algorithms that are able to produce signatures indistinguishable from a standard one. The protocol described in this work is, as far as we know, the first example of Schnorr threshold multi-signature allowing the presence of an offline participant during key-generation and whose signatures are indistinguishable from Schnorr ones.

The focus of this work was to shift away from DSA-like protocols, further motivated by the recent adoption of Schnorr signatures in Bitcoin[4]. Moreover, Schnorr signatures are quite a multi-party-friendly algorithm, unlike EdDSA, since we can avoid expensive tricks to generate a deterministic nonce.

Similarly to its ECDSA and EdDSA counterparts, in order to guarantee the security of the signature itself against black-box adversaries, the protocol involves a large utilization of ZKPs, that are the main bottleneck in terms of efficiency.

Future research steps could be the generalization to (t, n)-threshold schemes with more than one offline party and the extension of our notion of security. Although our protocol is susceptible to DOS attacks on the offline party, there are many ways to overcome this apparent weakness, such as the distribution of the role of the Recovery party to multiple servers or the generalization of our scheme to more than three parties.

Acknowledgments

The core results of this paper are contained in the Master's Thesis of the second author, supervised by the first and fourth authors.
The third and the fourth authors are members of the INdAM Research group GNSAGA.
The first author acknowledges support from TIM S.p.A. through the PhD scholarship.

References

[1] Michele Battagliola, Riccardo Longo, Alessio Meneghetti, and Massimiliano Sala. A provably-unforgeable threshold eddsa with an offline recovery party. CoRR, abs/2009.01631, 2020.

[2] Michele Battagliola, Riccardo Longo, Alessio Meneghetti, and Massimiliano Sala. Threshold ecdsa with an offline recovery party. Mediterranean Journal of Mathematics, 19(1):1–29, 2022.

[3] Mihir Bellare and Phillip Rogaway. Introduction to modern cryptography, 2005. https://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf.

[4] Dan Boneh, Rosario Gennaro, and Steven Goldfeder. Using level-1 homomorphic encryption to improve threshold dsa signatures for bitcoin wallet security, 2017.

[5] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. Journal of computer and system sciences, 37(2):156–189, 1988.

[4]See https://github.com/taprootactivation/Taproot-Activation, accessed 29th April 2022

[6] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, Advances in Cryptology - CRYPTO 2003, pages 126–144, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[7] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pages 1769–1787, 2020.

[8] Vincenzo Di Nicola. Custody at conio-part 3, 2020. https://medium.com/conio/custody-at-conio-part-3-623292bc9222.

[9] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Secure two-party threshold ecdsa from ecdsa assumptions. In 2018 IEEE Symposium on Security and Privacy (SP), pages 980–997. IEEE, 2018.

[10] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ecdsa from ecdsa assumptions: The multiparty case. In 2019 IEEE Symposium on Security and Privacy (SP), pages 1051–1066. IEEE, 2019.

[11] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In 28th Annual Symposium on Foundations of Computer Science (sfcs 1987), pages 427–438, 1987.

[12] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pages 1179–1194. ACM, 2018.

[13] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In International Conference on Applied Cryptography and Network Security, pages 156–174. Springer, 2016.

[14] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold dss signatures. In International Conference on the Theory and Applications of Cryptographic Techniques, pages 354–371. Springer, 1996.

[15] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In International Conference on the Theory and Applications of Cryptographic Techniques, pages 295–310. Springer, 1999.

[16] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In Proceedings of the 27th Annual Symposium on Foundations of Computer Science, pages 174–187, 1986.

[17] Yashvanth Kondi, Bernardo Magri, Claudio Orlandi, and Omer Shlomovits. Refresh when you wake up: Proactive threshold wallets with offline devices. IACR Cryptol. ePrint Arch., 2019:1328, 2019.

[18] Yehuda Lindell. Fast secure two-party ecdsa signing. In Annual International Cryptology Conference, pages 613–644. Springer, 2017.

[19] Yehuda Lindell and Ariel 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, pages

1837–1854. ACM, 2018.

[20] Philip MacKenzie and Michael K Reiter. Two-party generation of dsa signatures. In Annual International Cryptology Conference, pages 137–154. Springer, 2001.

[21] Philip MacKenzie and Michael K Reiter. Two-party generation of dsa signatures. International Journal of Information Security, 2(3-4):218–239, 2004.

[22] Antonio Marcedone and Claudio Orlandi. Obfuscation → (ind-cpa security ↛ circular security). In International Conference on Security and Cryptography for Networks, pages 77–90. Springer, 2014.

[23] Gregory Neven, Nigel P Smart, and Bogdan Warinschi. Hash function requirements for schnorr signatures. Journal of Mathematical Cryptology, 3(1):69–87, 2009.

[24] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Conference on the Theory and Application of Cryptology, pages 239–252. Springer, 1989.

[25] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In Annual International Cryptology Conference, pages 148–164. Springer, 1999.

[26] Adi Shamir. How to share a secret. Commun. ACM, 22(11):612613, November 1979.

[27] Victor Shoup and Joel Alwen. Σ-Protocols Continued and Introduction to Zero Knowledge, 2007. https://cs.nyu.edu/courses/spring07/G22.3220-001/lec3.pdf.