Efficient Threshold-Optimal ECDSA *

Michaella Pettit

nChain AG, Zug, Switzerland
m.pettit@nchain.com

Abstract. This paper proposes a threshold-optimal ECDSA scheme based on the first threshold signature scheme by Gennaro et al. with efficient non-interactive signing for any $t + 1$ signers in the group, provided the total group size is more than twice the threshold t. The scheme does not require any homomorphic encryption or zero-knowledge proofs and is proven to be robust and unforgeable with identifiable aborts tolerating at most t corrupted participants. The security of the scheme is proven in a simulation-based definition, assuming DDH and that ECDSA is existentially unforgeable under chosen message attack. To evaluate the performance of the protocol, it has been implemented in C++ and the results demonstrate the non-interactive signing phase takes 0.12ms on average meaning over 8000 signatures can be created per second. With pre-signing phase, it takes 3.35ms in total, which is over 144 times faster than the current state of the art.

Keywords: ECDSA · Multiparty computation · Threshold signatures

1 Introduction

A (t, N) threshold signature scheme is a method for a group of N participants to generate a signature on a message, without any individual participant having knowledge of the private key. A valid signature cannot be created by less than $t + 1$ participants. A benefit of using a threshold signature scheme is that the private key never exists at any point in time. There is no single point of failure, which mitigates against attack or loss of a private key.

One of the first threshold ECDSA schemes was proposed by Gennaro, Jarecki, Krawczyk, and Rabin [1]. A private key with threshold t is split between participants such that a subset of $2t+1$ participants are required to create a signature. This protocol is fast during signing, in which a participant can compute their share of the signature upon request without knowledge of other signers. This absence of back-and-forth communication is known as non-interactive signing.

The drawback of [1] is that the threshold of participants required to create a signature is $2t$, which is twice the threshold of computing of the private key. The multiplication of two shared secrets, each with threshold t, requires $2t + 1$ participants. In the context of [1] the two shared secrets are the private key and ephemeral key.

* An updated version of this paper has been accepted to CANS2021.

Further work focused on achieving threshold-optimality in which the private key and signing threshold are the same, initially for two signers [2–5]. In 2016, a scheme by Gennaro et al. [6] was the first to achieve threshold-optimality for any threshold t and group size N in theory. It required a distributed generation of an RSA modulus, which cannot efficiently involve more than two parties. As a consequence, [6] cannot achieve more than $(1, N)$ in practice.

The first practical (t, N) threshold-optimal scheme for any threshold t and group size N was published in 2018 by Gennaro and Goldfeder [7]. This scheme was based on [6] and achieves optimality by turning multiplication of two secrets into an addition of secrets using homomorphic encryption, along with zero-knowledge proofs to ensure security of the scheme. This leads to multiple rounds of communication and an increase in computation, particularly during signing. The signing protocol requires one-to-one communication with every other signing participant, limiting the scaling capability of the scheme. If a participant drops offline during signing, the signing protocol must be restarted.

Recently, there have been many (t, N) threshold-optimal schemes proposed [8–12]. Their use of homomorphic encryption and zero-knowledge proofs means that they still require expensive computation and interactive signing.

In 2020, Canetti et al. [13] and Gennaro and Goldfeder [14] each proposed a non-interactive threshold-optimal scheme, with the latter including identifiable abort. However, both schemes still rely on homomorphic encryption and zero-knowledge proofs. Another property of these schemes is that the participants who must collaborate during the non-interactive signing process is predetermined.

In spite of recent advances in threshold ECDSA, to the best of the author's knowledge none of the current schemes have achieved threshold-optimality without expensive computation such as homomorphic encryption and zero-knowledge proofs. Additionally, the signing must either be interactive or involve a set of participants that is decided before the message has been received, and either case results in a large number of rounds of communication and a high demand on overall computation.

Contributions. This paper proposes an efficient threshold-optimal ECDSA scheme.

- *Low computational complexity*: this is the first scheme to achieve threshold optimality without expensive computation like homomorphic encryption or zero-knowledge proofs on discrete logarithms, ranges of discrete logarithms, or others. Results show that it is over 144 times faster than [14] and almost 240 times faster than [13].
- *Low number of communication rounds*: the scheme requires four rounds in the signing protocol with identifiable abort where only the first requires secure one-to-one communication, equivalent to [1]. This is the same number of rounds as [13] and three rounds fewer than the protocol with identifiable abort in [14]. There are two rounds of communication in key generation which one round fewer than [14] and [13].

– *Non-interactive threshold-optimal signing*: the scheme is split into a pre-signing phase and a non-interactive signing phase once the message is known, similar to [14] and [13]. The signers are not predetermined in the signing step, unlike [14] and [13]. Therefore, any failures by less than $N - t$ participants does not affect the ability to complete the final round.
– *Identifiable corrupted participants*: participants that deviate from the protocol can be identified, in line with the recent proposal in [14].
– *Provably secure*: a simulation-based security proof is provided to show that the scheme is robust and unforgeable.

2 Preliminaries

2.1 Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman. Let $\mathcal{G}$ be a cyclic group of prime order n generated by G. The following are computationally indistinguishable: (aG, bG, abG) with $a, b \in_R \mathbb{Z}_n$ and (aG, bG, cG) with $a, b, c \in_R \mathbb{Z}_n$.

2.2 ECDSA

The Digital Signature Algorithm is a digital signature scheme proposed by Kravitz [15] in 1991.[1] The public parameters PP in the scheme are an elliptic curve group $\mathcal{G}$ with points over the finite field $\mathbb{F}_p$, generator G, and with order n.

– DSKeyGen: On input of a security parameter 1^l, this outputs a random private key $a \xleftarrow{\$} \mathbb{Z}_n^*$ and the corresponding public key $P = aG$ where aG is notation for point multiplication on an elliptic curve.
– DSSign: In order to calculate the signature on a message m using the private key a, the following steps are taken.
  1. Calculate the hash of the message $e \leftarrow hash(m)$.
  2. Randomly generate an ephemeral key $k \xleftarrow{\$} \mathbb{Z}_n^*$.
  3. Calculate $(x, y) \leftarrow kG$ then $r \leftarrow x \mod n$. If $r = 0$ return to Step 2.
  4. Calculate $s \leftarrow k^{-1}(e + ar) \mod n$. If $s = 0$ return to Step 2, otherwise output the signature as (r, s).
– DSVerify: In order to verify a signature (r, s) on a message m with a given public key P, the following steps are taken.
  1. Calculate the hash of the message $e \leftarrow hash(m)$.
  2. Calculate $(x', y') \leftarrow s^{-1}(eG + rP)$.
  3. Check if $r \stackrel{?}{=} x' \mod n$.

[1] The method can be applied to elliptic curve groups as given here, but it is understood that it may be applied to generic cyclic groups used in the standard DSA.

2.3 Threshold Signature Scheme

A threshold signature scheme is a tuple of protocols.

- TSKeyGen: The key generation algorithm takes public parameters PP as input. The output is composed of private outputs a_i known only to participant i for $i = 1, \ldots, N$, forming a (t, N) shared secret scheme corresponding to shared private key a, and a public output known to all participants which is the public key P corresponding to the shared private key.
- TSSign: The signing algorithm takes private key shares a_i and a message m in the message space $\mathcal{M}$ and outputs a signature sig.
- TSVerify: The verification algorithm has the public key P, signature sig, and message m as input, and outputs 1 if the signature is valid, or 0 otherwise.

2.4 Communication Model

In a scheme with N participants, it is assumed that they are connected by one-to-one secure communication channels and a broadcast channel. If participant i broadcasts a message, it is identifiable as being from that participant.

2.5 Adversary Model

It is assumed that an adversary can corrupt at most t participants in a threshold signature scheme, where $t + 1$ shares are required to reconstruct the private key. It is also assumed that the adversary has computational power that can be modelled by a probabilistic polynomial time (PPT) machine. There are three subtypes of adversaries:

- *Eavesdropping adversary*: this is a passive adversary that learns all information stored at corrupted nodes and all broadcasted messages.
- *Halting adversary*: this is an active adversary that is eavesdropping and may also stop corrupted participants from sending messages at each round of the protocol.
- *Malicious adversary*: this is an active adversary that may cause any corrupted participant to deviate from the protocol.

A halting or malicious adversary may also be a *rushing adversary*, which is one that ensures corrupted participants speak last in any rounds of communication and may reorder any messages that are sent.

Definition 1. *As defined in* [1]*, the* view of the adversary *is the knowledge of the adversary in a protocol. That is, the computational history of all corrupted participants and public communications, including the output of the protocol.*

The definitions of unforgeability and robustness are now given. These will enable a secure threshold signature scheme to be defined.

Definition 2. *A (t, N) threshold signature scheme is* unforgeable *if no malicious PPT adversary can produce a valid signature on a previously unsigned message m with non-negligible probability, where the adversary has knowledge of the following: the output of the key generation protocol $a_1, \ldots, a_t$ and P, and the output of the signature generation protocol $sig_1, \ldots, sig_\nu$ on messages $m_1, ..., m_\nu$, which the adversary chose.*

Definition 3. *A threshold signature scheme is* robust *if* TSKeyGen *and* TSSign *produce the expected outputs even in the presence of a halting or malicious adversary. An expected output of* TSKeyGen *is one in which a_i for $i = 1, \ldots, N$ are shares of a (t, N) shared secret that corresponds to the public output P. For* TSSign*, an expected output is one that is accepted by verification using* TSVerify*.*

For robustness, it does not matter if more than t participants are corrupted by an eavesdropping adversary, the protocol will still produce an expected output.

Definition 4. *A (t, N) threshold signature scheme is* secure *if it is robust and unforgeable in the presence of an adversary who corrupts at most t participants.*

In order to prove the unforgeability of the threshold scheme, it is necessary to be able to simulate the scheme. This is the definition from [1].

Definition 5. *A threshold signature scheme is* simulatable *if:*

1. *The key generation protocol* TSKeyGen *is simulatable. That is, there exists a simulator that can simulate the view of an adversary in an execution of* TSKeyGen *given the input of the public key and the public output generated by an execution of* TSKeyGen*.*
2. *The signing protocol* TSSign *is simulatable. That is, there exists a simulator that can simulate the view of the adversary on an execution of* TSSign *that takes the public key, message, t shares of a shared private key, and the signature on the message as input, and generates sig as an output.*

The security is proven by comparing the view of the adversary in the protocols TSKeyGen and TSSign to an ideal setting. This ideal setting is a simulation that is secure by definition. Therefore, showing that the view is indistinguishable to the attacker proves that the protocols TSKeyGen and TSSign are secure.

2.6 Verifiable Random Secret Sharing [16]

The TSKeyGen and TSSign protocols require a (t, N) secret sharing protocol, which has been chosen to be the scheme in [16] and has two rounds of communication.

– VRSS: This is the shared secret generation algorithm that takes the index i of each participant and the threshold t as input and outputs a share a_i of a shared secret for each participant i.

1. Each participant i randomly generates integers $a_{il}, b_{il} \xleftarrow{\$} \mathbb{Z}_n$ for $l = 0, \ldots, t$, where a_{il}, b_{il} are the coefficients for the degree-l term in the polynomials $f_i(x)$ and $f'_i(x)$, respectively. Each participant i computes and broadcasts $C_{il} = a_{il}G + b_{il}H$ for each l, where H is a generator of the group and it is assumed an adversary cannot compute $log_G H$. Each participant i sends $f_i(j)$, $f'_i(j)$ via a one-to-one communication channel to participant j for each $j \neq i$.
2. Each participant j verifies if $f_i(j)G + f'_i(j)H \stackrel{?}{=} \sum_{l=0}^{t} (j)^l C_{il}$, for all $i \neq j$. If any i fails, participant j broadcasts a complaint against participant i.
3. Each participant i who was the subject of a complaint in the previous step broadcasts the values $f_i(j)$, $f'_i(j)$ satisfying the equation in Step 2.
4. The set of non-disqualified parties Q are those that received t or fewer complaints in Step 2, or answered the complaints with correct values.
5. Each participant $i \in Q$ calculates their secret share $a_i \leftarrow \sum_{j \in Q} f_j(i)$.
6. Each participant $i \in Q$ calculates and broadcasts their obfuscated coefficients $a_{il}G$ for each l.
7. Each participant j verifies if $f_i(j)G \stackrel{?}{=} \sum_{l=0}^{t} (j)^l (a_{il}G)$, for all $i \neq j$. If any i that passed the check in Step 2 fails this verification, participant j broadcasts a complaint against that participant by sharing the values $f_i(j)$, $f'_i(j)$ they received.
8. For each participant i reconstructs the values a_{j0} and $a_{j0}G$ for each participant j who receives a valid complaint, that is, those values that satisfy the equation in Step 2 and not in Step 7. Each participant constructs $P \leftarrow \sum_{i \in Q} a_{i0}G$.

Shares a_i allow operations on the shared secret values to be computed whilst keeping the value of the shared secrets hidden, even to the participants of the scheme. That is, the shared secret values never exist and cannot be computed by any participant unless the threshold is passed. Note that Feldman's verifiable secret sharing scheme [17] can allow an adversary to change the distribution of the public key and so can be used for shared secrets in which the corresponding public key is fixed or not used in the scheme, or if the corrupted participants are eavesdropping only.

2.7 Verifiable Zero Secret Sharing [1]

It will be necessary to create shares of the value zero using (t, N) verifiable zero secret sharing VZSS. This uses Feldman's verifiable secret sharing scheme [17] as the corresponding public key is fixed, meaning an adversary cannot change the distribution.

– VZSS: This is the shared secret generation algorithm that takes the index i of each participant and the threshold t as input and outputs a share a_i of a zero-valued shared secret for each participant i.

1. Each participant i randomly generates integers $a_{il} \xleftarrow{\$} \mathbb{Z}_n$ for $l = 1, \ldots, t$ and sets $a_{i0} \leftarrow 0$, where a_{il} is the coefficient for the term of degree l in the polynomial $f_i(x)$. Each participant i sends $f_i(j)$ via a one-to-one communication channel to participant j for each $j \neq i$.
2. Each participant i calculates their secret share $a_i \leftarrow \sum_{j=1}^{N} f_j(i)$.
3. Each participant i calculates and broadcasts their obfuscated coefficients $a_{il}G$ for each $l = 1, \ldots, t$.
4. Each participant j calculates $f_i(j)G$ using the value received in Step 2 and verifies if $f_i(j)G \stackrel{?}{=} \sum_{l=1}^{t}(j)^l(a_{il}G)$, for all $i \neq j$. Participant j broadcasts a complaint for any participant i which values do not satisfy this equation.

By adding zero-shares to computations with shared secrets, a randomization of the shares is achieved without changing the result of the computation.

2.8 Operations on Shared Secrets

Given multiple shared secrets where the shares are points on a polynomial, it is possible to directly compute operations such as addition of secrets, multiplication of secrets, multiplication by a constant, or a combination of these simultaneously, provided enough shares of each shared secret are available. The shares k_i^{-1} that correspond to the inverse of a (t, N) shared secret with shares k_i are computed using the following protocol as given in [1].

– SSInverse: This takes shares k_i for $i = 1, \ldots, N$ as input and outputs the corresponding inverse shares k_i^{-1} for each i.
  1. All participants execute a (t, N) shared secret scheme, where the share of participant i is denoted by α_i.
  2. Each participant i computes $\mu_i \leftarrow \alpha_i k_i$ and broadcasts the result.
  3. All participants calculate $\mu \leftarrow interpolate(\mu_1, \ldots, \mu_{2t+1})$, where the notation $interpolate(\ldots)$ is Lagrange interpolation evaluated at $x = 0$ over shares $\mu_1, \ldots, \mu_{2t+1}$.
  4. Each participant i calculates their inverse share $k_i^{-1} \leftarrow \mu^{-1}\alpha_i$.

3 Efficient Threshold-Optimal Scheme

Threshold ECDSA signature generation involves the multiplication of two shared secrets, each with a threshold of t. The present scheme illustrates that it is possible to precalculate all multiplications prior to receiving a message without the use of expensive computation. The signature generation on the message is threshold-optimal and non-interactive, with no restriction on the $t + 1$ participants that sign. While the number of participants required to calculate the multiplication in precalculation is $2t + 1$, the signature threshold during the non-interactive signing phase is now the same as the threshold t of the private key.

Observe that a signature in ECDSA has the form (r, s), where

$$s = k^{-1}e + k^{-1}ar \ .$$

Here, e is the hash of the message, a is the private key, and r is derived from the public key corresponding to the ephemeral key k. The second term is independent of the message, meaning that it can be calculated prior to receiving a message in a pre-signing phase. However, if the value $k^{-1}a$ itself is known, as soon as the signature is calculated it is trivial to calculate the private key a. To secure the result $k^{-1}a$, another (t, N) shared secret β is added into this computation.

Explicitly, the signature is

$$s = k^{-1}e + r(\sigma - \beta)$$

where $\sigma = k^{-1}a + \beta$ is precalculated. The signature is now an addition of k^{-1} and β which are both (t, N) shared secrets, therefore only $t + 1$ shares are required.

While at least $2t+1$ participants are required to execute the complete scheme, during the final step once the message is known the number of participants required $t + 1$ is the same as the number required to calculate the private key and this may be any subset of the group. Therefore, threshold-optimality is achieved in the non-interactive signing phase without requiring expensive computations like homomorphic encryption or zero-knowledge proofs. Because of the absence of these expensive computations, it is feasible for multiple k, σ, and corresponding r values to be precalculated in parallel and stored until required. Benchmarking shows an average time of this pre-computation in a scheme with three particiapants, 2 of which are required to compute a signature, is 3.22ms before any parallelisation, meaning that over 310 values could be calculated per second. One of these precalculated values is used with each new signature and then discarded.

If optimised, the rounds of communication may be as low as three prior to receiving the message and there will be only one round during signature generation. Similarly, after the initial round during VRSS, which has one-to-one communication, the remaining rounds are broadcasts, including signature generation. The implication of this is that the scheme is easily scalable.

3.1 Distributed Key Generation

The following protocol is a known result. TSKeyGen takes the public parameters PP as input and outputs a secret share a_i only known to participant i that corresponds to a share of a (t, N) shared private key a and a public output that is the public key P. The protocol has two rounds of communication since it uses VRSS. Assume all participants have agreed on each other's unique, non-zero integer i, usually chosen to be $i = 1, \ldots, N$.

TSKeyGen
Input: public parameters PP, index i for $i = 1, \ldots, N$, threshold t
Output: shares a_i for $i = 1, \ldots, N$, public key P

1. All participants execute a (t, N) shared key generation VRSS where participant i obtains the secret output a_i and public output P.

At the end of this protocol, each participant i stores a share a_i and the public key P where the public key $P = aG$ is the same for all participants.

3.2 Signature Generation

The signature generation protocol TSSign allows for precalculation which has 3 rounds of communication. The participants compute all the possible values that are independent of the message and store until it is required to calculate a signature on a message m in the final round.

TSSign
Input: private key shares a_i for $i = 1, \ldots, N$, message m
Output: signature (r, s)

1. All participants calculate the ephemeral key shares and corresponding public key using a (t, N) execution of VRSS, where participant i's share is k_i and the public key is $(x, y) \leftarrow \sum_{i=1}^{N} k_{i0}G$. All participants calculate $r \leftarrow x \mod n$.
2. All participants create two (t, N) shared secrets using two instances of VRSS with resulting shares denoted by α_i and β_i corresponding to participant i. Each participant i also calculates the commitment of α_j and β_j

$$\alpha_j G \leftarrow \sum_{l=1}^{N} \sum_{m=0}^{t} j^m (\alpha_{lm} G) \ ,$$
$$\beta_j G \leftarrow \sum_{l=1}^{N} \sum_{m=0}^{t} j^m (\beta_{lm} G) \ ,$$

   for each participant $j \neq i$, where $\alpha_{lm}G$ and $\beta_{lm}G$ are received during Step 6 of VRSS, and stores $\alpha_j G$ and $\beta_j G$.
3. All participants create a zero-valued $(2t, N)$ shared secret with shares denoted by κ_i for participant i using VZSS.
4. Each participant i calculates $\mu_i \leftarrow \alpha_i k_i + \kappa_i$ and $\lambda_i \leftarrow \alpha_i a_i + \beta_i$, and $\alpha_i(kG)$, $(\alpha_i P + \beta_i G)$ and broadcasts these.
5. Each participant i verifies

$$interpolate(\mu_i, \ldots, \mu_{i'})G \stackrel{?}{=} interpolate(\alpha_i(kG), \ldots, \alpha_{j'}(kG)) \ ,$$
$$interpolate(\lambda_i, \ldots, \lambda_{i'})G \stackrel{?}{=} interpolate((\alpha_i P + \beta_i G), \ldots, (\alpha_{j'} P + \beta_{j'} G)) \ ,$$

   where $i' = (i + 2t + 1)$ and $j' = (i + t + 1)$. If the index i' is larger than N, the values wrap around to index 1 again. If any of these are found to be different, the adversaries are identified by interpolating over all possible sets of shares and all sets which result in the same values contain only honest participants. Corrupted participants are identified as those not contained in these sets.

6. If the equalities hold for all participants, each participant i sets

$$\mu \leftarrow interpolate(\mu_1, \ldots, \mu_{2t+1}) \quad (= \alpha k) \ ,$$
$$\lambda \leftarrow interpolate(\lambda_1, \ldots, \lambda_{2t+1}) \quad (= \alpha a + \beta) \ .$$

7. Each participant i calculates their inverse shares $k_i^{-1} \leftarrow \mu^{-1}\alpha_i$ of the shared ephemeral key and their precalculated shares $\sigma_i \leftarrow r\mu^{-1}(\lambda - \beta_i)$.
8. Each participant i stores (r, k_i^{-1}, σ_i) for use in the signature computation and $(\alpha_j G, \beta_j G)$ for all participants j for verification of the signature.

The pre-signing phase can be executed prior to receiving any message. The non-interactive signing phase takes the message m and precalculated values (r, k_i^{-1}, σ_i) as input and output the signature (r, s).

9. At least $t + 1$ participants compute the hash the message $e = hash(m)$, calculate their signature share $s_i \leftarrow k_i^{-1}e + \sigma_i$, and broadcast.
10. Participants set $s \leftarrow interpolate(s_1, \ldots, s_{t+1})$ and the signature is (r, s).

A signature has been computed using only $t+1$ shares after precalculation. Note that TSVerify is the same as DSVerify described in Section 2.2 and is not repeated here. If the signature is found to be incorrect using TSVerify, the corrupted participants are identified using CorruptID.

3.3 Identifiable Abort

If the signature generated with TSSign is found to be incorrect, the following protocol is executed. Assume the participants that have signed are those with indices $i = 1, \ldots, t + 1$, without loss of generality.

CorruptID
Input: obfuscated shares $\alpha_i G, \beta_i G$ for $i = 1, \ldots, t + 1$
Output: identity of corrupted participants j

1. Each participant i calculates $k_j^{-1}G \leftarrow (k\alpha)^{-1}\alpha_j G$ and $\sigma_j G \leftarrow r\mu^{-1}(\lambda G - \beta_j G)$ for each participant j who executed Step 9 to 10 in TSSign.
2. Each participant then checks $s_j G \stackrel{?}{=} e(k_j^{-1}G) + (\sigma_j G)$ for each j. If this does not hold for a given j, that share is incorrect.

3.4 Discussion

In TSSign, a signature has been created with the same threshold as that of the shared private key, after the precalculation steps have been completed. The s value of the signature can be written as

$$s = \mu^{-1}\alpha e + r\mu^{-1}(\lambda - \beta) \ ,$$

where α and β are (t, N) shared secrets, and $\mu = \alpha k$ and $\lambda = \alpha a + \beta$ are precalculated. By replacing μ and λ, this becomes $s = k^{-1}(e + ar)$, as required.

The computation of λ may be considered a method to calculate the multiplication of two shared secrets whilst hiding the result. With each shared secret having a threshold t, the computation of λ requires $2t+1$ shares. Interpolation over $t+1$ shares of σ_i will result in $k^{-1}a$ as the β terms will cancel. This share σ_i may be seen as a share of $k^{-1}a$ with a threshold of t. Therefore, after Step 6 the threshold of the multiplication of the two shared secrets is reduced to t.

While $2t+1$ participants are required until Step 8 of TSSign, after this only $t+1$ of the N participants are required. There may be multiple simultaneous computations running up to Step 8, at which point these σ_i, k_i^{-1}, r values may be stored until required for use. Once a value is used for signing, these are not used again. No expensive computation such as homomorphic encryption or zero-knowledge proofs are required as all previous threshold-optimal constructions [6–14, 18]. Instead of homomorphic encryption and zero-knowledge proofs, this protocol has three additional executions of VRSS compared to other threshold-optimal protocols, however it will be shown to be 144 times faster than [14] which also proposes precomputation.

Calculation and verification of $\alpha_i(kG)$ and $(\alpha_i P + \beta_i G)$ in Steps 4 and 5 ensure robustness. Without these, it would be possible for an unidentifiable corrupted participant j to send incorrect values for μ_j or λ_j, which would prevent a valid signature being created. If a corrupted participant attempts to send an incorrect value for the multiplication of their shares, there is no value that will pass the verifications aside from the correct value. This is because every participant is interpolating over a different set of shares and knows that all participants receive the same broadcasted values. The same logic can be applied to the verification of λ_i and $\alpha_i P$.

Note that in the non-interactive signing phase of the scheme, the signature is calculated assuming the participants are honest, but the result is verified for correctness. If the signature is found to be invalid, the shares are checked individually to identify the incorrect share. This does not require any further rounds of communication, since all participants already have enough knowledge to verify shares. While this could be executed prior to calculating the signature, it would slow down those rounds which are executed correctly and is therefore more efficient to perform these verification steps only if necessary.

4 Security Proof

In this section, the following theorem is proven, assuming it is infeasible to forge a signature in ECDSA [19].

Theorem 1 *The threshold signature scheme in Section 3 is secure in the presence of d participants corrupted by an eavesdropping adversary and h participants corrupted by a halting or malicious adversary, if the total number of participants is $N > 2t + h$ and number of corrupted participants is $d + h \leq t$.*

The proof is split into proving robustness and then unforgeability.

Lemma 2 *The threshold signature scheme in Section 3 is robust, if the number of participants in the scheme is $N > 2t+h$ where h is the number of participants corrupted by a halting (or malicious) adversary.*

Proof. The scheme will be shown to be robust in the presence of $2t + 1$ participants who do not deviate from the protocol, such that the signature that is generated will always be accepted by an execution of TSVerify. Note that these participants may include d eavesdropping participants. Shares that belong to participants who do not deviate from the protocol will be referred to as correct shares. Specifically, the requirement of $2t + 1$ correct shares is due to the multiplications of (t, N) shared secrets in Step 4 of TSSign. After this point, only $t + 1$ correct shares are required to be robust.

There are three rounds of communication in which there is scope for participants to send values that deviate from the protocol. Each of these rounds are followed by verifications and it is these steps that identify correct shares.

- Step 2 and Step 7 of VRSS executed in TSKeyGen and TSSign: if shares of any $f_i(j)$ are not received or invalid before Step 4, all shares of that private polynomial are removed from the calculation. The values that do not validate correctly in Step 7 are recovered by the honest participants using the values received in Step 1 to compute the public key corresponding to the shares. There will be at most h private polynomials removed, with at least $2t + 1$ remaining. Therefore, there will still be at least $2t + 1$ shares a_i output from VRSS due to the requirement of $N > 2t + h$.
- Step 5 of TSSign: there must be $2t + 1$ shares of μ or λ, since they are a multiplication of (t, N) shared secrets. This is the case as there are at least $2t + 1$ participants who do not deviate from the protocol. These shares can be detected as those which are contained in sets that find the equalities in this step hold. Therefore, enough correct shares exist and can be identified. The calculations of μ and λ use these correct shares.
- Steps 1 and 2 in CorruptID: the shares s_i which agree with the obfuscated shares calculated from the execution of VRSS are used in the computation of the signature. Since there are at least $2t + 1$ participants that do not deviate from the protocol, and there are $t + 1$ required for this calculation, there will always enough shares to calculate the signature. These shares can be detected using CorruptID.

It has been illustrated that incorrect shares can always be detected and there will always be enough correct shares remaining for each computation. Hence, TSKeyGen and TSSign will produce expected outputs given $N > 2t + h$ and the scheme described in Section 3 is robust. □

The proof of unforgeability is given by proving each protocol can be simulated in a way which the adversary cannot distinguish the simulation from the real protocol. In order to prove that TSSign is indistinguishable from its simulation, it is necessary to understand how to generate an elliptic curve point (x, y) from the r value in a signature such that the point appears uniformly random among

the set of all candidates of (x, y). Recall that the r value of a signature (r, s) is an x value of an elliptic curve point modulo n and the order of the field that the elliptic curve is defined over is p.

ECPointDerivation
Input: r
Output: $\hat{k}G$

1. To calculate the x value
   - If $n \geq p$, set $x \leftarrow r$.
   - If $n < p$ and
     - $r \geq (p - n)$, set $x \leftarrow r$.
     - $r < (p - n)$, calculate $(r + n) \mod p$. Check which of r and $(r + n)$ correspond to x values on the elliptic curve.
       * If only one is an x value on the elliptic curve, set that to x.
       * If both correspond to an x value, randomly select one and set it to x.
2. Calculate a y' value corresponding to x according to the elliptic curve equation. If there is only one unique y' value, set $y \leftarrow y'$, otherwise calculate $-y'$ and randomly select $y \leftarrow y'$ or $y \leftarrow -y'$.
3. Set the point to be $\hat{k}G \leftarrow (x, y)$.

This derivation ensures that the distribution of points kG remains uniform when derived from r. The point that r was derived from does not need to be the same as the point that is found with this method.

Each protocol in the scheme described in Section 3 is now shown to be simulatable and indistinguishable from that simulation. In each step of the simulations, the action in the brackets describe the steps the adversary takes. It is assumed that the adversary generates the values corresponding to the corrupted participants. This is stronger than assuming that the adversary only learns the shares of the corrupted participants and so subsumes this case. Note the steps described as 'twiddle thumbs' are paying homage to [1] and are used where the protocol requires the simulator to do nothing during that step.

VRSS is proven to be secure in [16], which is used in both the key generation simulation and in the signature generation simulation, denoted VRSS-sim. The input to the simulation is the public key P, indices i, and threshold t.

Lemma 3 *The* TSKeyGen *protocol described in Section 3.1 is simulatable and is indistinguishable from its simulation from the point of view of the adversary.*

Proof. Assume that the indices i of participants have been generated already and, without loss of generality, that the adversary has corrupted participants $i = 1, \ldots, t$. The steps in the following simulation coincide with the steps in the protocol in Section 3.1.

TSKeyGen-sim
Input: public key P, index i for $i = 1, \ldots, N$, threshold t
Output: shares $\hat{a}_i$ for $i = 1, \ldots, N$, public key P

1. The simulator invokes VRSS-sim outputting a_i for $i = t+1, \ldots, N$ and P. (The adversary executes VRSS to calculate a_i for $i = 1, \ldots, t$ and P.)

It has been shown in [16] that VRSS is indistinguishable from VRSS-sim. The signature that is to be generated can be verified against this public key and the verification will be accepted. These steps are therefore indistinguishable to the adversary from TSKeyGen. □

In order to simulate TSKeyGen, VZSS is first shown to be unforgeable.

Lemma 4 *The* VZSS *protocol described in Section 2.7 is simulatable and is indistinguishable from its simulation from the point of view of the adversary.*

Proof. The simulation of VZSS is given below.

VZSS-sim
Input: index i, threshold t
Output: shares $\hat{a}_i$ for $i = 1, \ldots, N$

1. The simulator generates uniformly random values $\eta_{ji} \in \mathbb{Z}_n^*$ for $j = t+1, \ldots, N$ and $i = 1, \ldots, t$ and shares $\hat{f}_j(i) \leftarrow \eta_{ji}$ with the adversary and receives $\hat{f}_i(j)$ for $j = t+1, \ldots, N$. (The adversary generates coefficients $\hat{a}_{il}$ of $\hat{f}_i(x)$ for $i, l = 1, \ldots, t$, shares $\hat{f}_i(j)$ and receives $\hat{f}_j(i)$ from the simulator for $j = t+1, \ldots, N$.)
2. Twiddle thumbs. (The adversary calculates $\hat{a}_i$ for $i = 1, \ldots, t$.)
3. The simulator calculates

$$\hat{f}_j(x) \leftarrow \sum_{l=1}^{t} \hat{f}_j(l) \prod_{\substack{1 \le j \le t+1, \\ j \ne i}} (x-j)(l-j)^{-1} \mod n \,,$$

   that satisfy the above values. The simulator uses these values to calculate $\hat{a}_j$, $\hat{a}_{jl}G$ for $j = t+1, \ldots, N$ and stores $\hat{a}_j$. The simulator shares $\hat{a}_{jl}G$ and receives $\hat{a}_{il}G$ from the adversary. (The adversary shares $\hat{a}_{il}G$ and receives $\hat{a}_{jl}G$ from the simulator.)
4. Twiddles thumbs. (The adversary verifies $f_i(j)$.)

In Step 1 above, the adversary receives shares $\hat{f}_i(j)$ from the simulator which are randomly generated and therefore uniformly distributed. Compare this to VZSS, where the adversary receives shares $f_i(j)$ which are calculated from the addition of values which are uniformly distributed. To an adversary, these sets of values are indistinguishable.

Similarly, in Step 3 of VZSS-sim, the adversary receives coefficients $\hat{a}_{jl}G$ which are calculated from the addition of randomly generated values η_{ji}, hence are uniformly distributed across the set. This ensures they are indistinguishable from the values $a_{jl}G$ that are received in VZSS.

Finally, the verifications in Step 4 will be accepted by the adversary due to the way that the coefficients are generated in Step 3 of the simulation. □

Lemma 5 *The* TSSign *protocol described in Section 3.2 is simulatable and is indistinguishable from its simulation from the point of view of the adversary.*

Proof. The steps in the following simulation coincide with those in the protocol in Section 3.2.

TSSign-sim
Input: shares $\hat{a}_1, \ldots, \hat{a}_t$, public key P, message m, signature (r, s)
Output: $\perp$

1. The simulator executes ECPointDerivation that outputs $\hat{k}G \leftarrow (x, y)$ with r as input. The simulator invokes VRSS-sim using input $\hat{k}G$, outputting $\hat{k}_i$ for $i = 1, \ldots, t$. (The adversary executes VRSS to obtain $\hat{k}_i$, $\hat{k}G$ and computes r.)
2. The simulator randomly generates $\hat{\alpha}$, $\hat{\beta}$, $\hat{\mu}$ and calculates $\hat{\lambda} \leftarrow r^{-1}(\hat{\mu}s - \hat{\alpha}e + \hat{\beta}r) \mod n$, where $e = hash(m)$. The simulator executes two instances of VRSS-sim to calculate $\hat{\alpha}_i$ and $\hat{\beta}_i$ for $j = 1, \ldots, N$, using $\hat{\alpha}$ and $\hat{\beta}$ as input. (Adversary calculates $\hat{\alpha}_i$ and $\hat{\beta}_i$ for $i = 1, \ldots, t$ using two instances of VRSS, $\hat{\alpha}_j G, \hat{\beta}_j G$ and stores.)
3. The simulator executes VZSS-sim outputting $\hat{\kappa}_i$ for $i = 1, \ldots, t$. (The adversary calculates $\hat{\kappa}_i$ using VZSS for $i = 1, \ldots, t$.)
4. The simulator takes the following steps.
   - Calculate $\hat{\mu}_i \leftarrow \hat{\alpha}_i \hat{k}_i + \hat{\kappa}_i$ for $i = 1, \ldots, t$. Calculate $\hat{\mu}_i$ for $i = t+1, \ldots, 2t$ such that $\hat{\mu}_i$ for $i = 1, \ldots, 2t$ defines a polynomial $\hat{f}(x)$ such that $\hat{f}(0) = \hat{\mu}$. Calculate $\hat{\mu}_i \leftarrow \hat{f}(i)$ for $i = 2t + 1, \ldots, N$.
   - Calculate $\hat{\lambda}_i \leftarrow \hat{\alpha}_i \hat{a}_i + \hat{\beta}_i$ for $i = 1, \ldots, t$. Calculate share $\hat{\lambda}_i$ for $i = t + 1, \ldots 2t$ such that $\hat{\lambda}_i$ for $i = 1, \ldots, 2t$ define a polynomial $\hat{g}(x)$ such that $\hat{g}(0) = \hat{\lambda}$. Calculate $\hat{\lambda}_i \leftarrow \hat{g}(i)$ for $i = 2t + 1, \ldots, N$.
   - Calculate $\hat{\alpha}_i(\hat{k}G)$, and $(\hat{\alpha}_i P + \hat{\beta}_i G)$ for $i = 1, \ldots, t$, and calculate $\hat{\mu}G$ and $\hat{\lambda}G$. Compute

$$\hat{\alpha}_j(\hat{k}G) \leftarrow interpolate(\hat{\mu}G, \hat{\alpha}_1(\hat{k}G), \ldots, \hat{\alpha}_t(\hat{k}G)) ,$$
$$(\hat{\alpha}_j P + \hat{\beta}G) \leftarrow interpolate(\hat{\lambda}G, (\hat{\alpha}_1 P + \hat{\beta}_1 G), \ldots, (\hat{\alpha}_t P + \hat{\beta}_t G)) ,$$

     for $j = t + 1, \ldots, N$ and where $\hat{\mu}G$ and $\hat{\lambda}G$ are the points at $x = 0$.
   - The simulator broadcasts $\hat{\mu}_i$, $\hat{\lambda}_i$, $\hat{\alpha}_i(\hat{k}G)$, and $(\hat{\alpha}_i P + \hat{\beta}_i G)$ for each $i = t+1, \ldots, N$ and receives values for $i = 1, \ldots, t$. (The adversary calculates $\hat{\mu}_i$, $\hat{\lambda}_i$, $\hat{\alpha}_i(\hat{k}G)$, and $(\hat{\alpha}_i P + \hat{\beta}_i G)$ and broadcasts.)
5. Twiddle thumbs. (The adversary verifies $\hat{\mu}_i$, $\hat{\lambda}_i$ and $\hat{\alpha}_i(\hat{k}G)$, $(\hat{\alpha}_i P + \hat{\beta}_i G)$ interpolate to the same result.)
6. Twiddle thumbs. (The adversary sets values $\hat{\mu}, \hat{\lambda}$.)
7. Calculate $\hat{k}_i^{-1} \leftarrow \hat{\mu}^{-1}\hat{\alpha}_i$ and $\hat{\sigma}_i \leftarrow r\hat{\mu}^{-1}(\hat{\lambda} - \hat{\beta}_i)$ for all $i = t + 1, \ldots, N$ given $\hat{\alpha}_i$, $\hat{\beta}_i$ calculated in Step 2. (The adversary calculates $\hat{k}_i^{-1}$ and $\hat{\sigma}_i$ for $i = 1, \ldots, t$.)
8. Twiddle thumbs. (The adversary stores values.)

9. The simulator calculates $\hat{s}_i \leftarrow \hat{k}_i^{-1}e + \hat{\sigma}_i$ for $t+1$ randomly selected values of i within the range $i = t+1, \ldots, N$ and shares these values. (The adversary calculates $\hat{s}_i$ for $i = 1, \ldots, t$.)
10. Twiddle thumbs. (The adversary calculates (r, s).)

While the signature (r, s) will be accepted if verified with the public key, the adversary can still ensure the shares generated by the simulator are also correct.

CorruptID-sim
Input: shares $\hat{s}_1, \ldots, \hat{s}_{t+1}$, message m
Output: $\perp$

1. Twiddle thumbs. (The adversary calculates $\hat{k}_j^{-1}G$ and $\hat{\sigma}_j G$ for each participant j that took part in the signature.)
2. Twiddle thumbs. (The adversary verifies that these obfuscated values were used to generate the signature shares $\hat{s}_i$ by comparing to $\hat{s}_i G$.)

In Step 1 to 3, the simulation of VRSS is used multiple times. It has been already shown that the simulation is indistinguishable from VRSS itself in [16]. Moreover, the public key $\hat{k}G$ that is calculated by the adversary in Step 1 is uniformly distributed across the set of elliptic curve points as it uses ECPoint-Derivation. Therefore, the first three steps are indistinguishable to the adversary from the first three steps in TSSign.

The values $\hat{\mu}$, $\hat{\alpha}$, $\hat{\beta}$ are randomly generated from a uniformly distributed set of values. All values in Step 4 of the simulation are derived from these, including $\hat{\mu}_i$, $\hat{\lambda}_i$, $\hat{\alpha}_i(\hat{k}G)$, and $(\hat{\alpha}_i P + \hat{\beta}_i G)$, which the adversary receives. Therefore, the values that the adversary receives also appear uniformly distributed. On the other hand, in TSSign, the corresponding values μ_i, λ_i, $\alpha_i(kG)$, and $(\alpha_i P + \beta_i G)$ that an adversary receives are similarly uniformly distributed, by the same reasoning. Therefore, an adversary will not be able to distinguish between the two sets.

Note that the values $\hat{\mu}$ and $\hat{\lambda} - \hat{\beta}$ are not equivalent to $\hat{\alpha}\hat{k}$ and $\hat{\alpha}a$. If they were, a and $\hat{k}$ could be revealed, since the values $\hat{\alpha}$, $\hat{\beta}$ must be known to ensure that $\hat{s}_i$ are accepted in CorruptID-sim. This contradicts the assumption that ECDSA is unforgeable. As a result of this, the DDH assumption is required as described in Section 2.1, similar to [1]. However, due to the construction of the values, the verifications by the adversary are still accepted in the simulation.

In Step 5 to 9, the adversary is executing their own calculations. In Step 10 the simulator shares the values $\hat{s}_i$. Since $\hat{s}_i$ are calculated from values that are uniformly distributed themselves, the result is that the set of signature shares are also uniformly distributed. Again, this is the same as the protocol TSSign and so the adversary will have the same view within the two protocols. The calculations executed by the simulator in Step 4 ensure that the shares will result in the correct signature. Step 4 also ensures that the signature shares will individually pass the checks in CorruptID-sim, as stated previously.

As a result, the calculations executed by the adversary will be accepted and have the same probability distribution. Therefore, the adversary will not be able to identify that it is in the simulation. □

Lemma 6 *The view of an adversary in the protocol described in Section 3 is indistinguishable from the view of the adversary in a simulation from the point of view of the adversary.*

Proof. It has been shown that both TSKeyGen and TSSign are simulatable and indistinguishable from their simulation in Lemma 3 and Lemma 5. Therefore, the view of the adversary in the scheme described in Section 3 is indistinguishable from the simulation from the point of view of the adversary. □

Lemma 7 *The threshold signature scheme described in Section 3 is unforgeable if the total number of participants is $N > 2t + h$ and the number of corrupted participants is $d + h \leq t$, where d and h are the number of participants corrupted by an eavesdropping adversary and by a halting (or malicious) adversary, respectively.*

Proof. If the total number of corrupted participants is more than t, that the (t, N) shared secret can be calculated and therefore also a signature. On the other hand, by assuming that $d + h \leq t$, the shared private key, and therefore signature, cannot be calculated. This has been shown in that the view of the adversary in TSKeyGen and TSSign is indistinguishable from their simulations. Since the simulations are unforgeable by definition, this means that TSKeyGen and TSSign are also unforgeable. □

5 Benchmarking

TSKeyGen and TSSign have been implemented assuming there are only eavesdropping adversaries in TSSign. That is, it uses Feldman secret sharing as in [17] and excludes Step 5 of TSSign and CorruptID. The implementation is compared data given for schemes [14] and [13] which are also non-interactive. The implementation was written in C++ and was run on a 2018 MacBook Pro with a 2.6 GHz Intel Core i7 processor and 32GB RAM. Participants were run as separate processes on a single machine using a single core. In practice, calculation by different participants is parallelizable and so the timings will be reduced further. The data for [14] is chosen to be the scheme without identifiable abort to compare fairly with the implementation of the scheme in Section 3. Also, the data for [13] was only available for precalculation of the signature and up to $t = 8$.

The scheme was run 20 times for each threshold and group size up to $t = 9$ and $N = 20$ in line with [14] and [13]. TSSign was split to measure the average time for the precalculation in Step 1 to 8, and the average time for the non-interactive signing in Step 9 to 10. Even if there are failures by at most t participants, this does not impact the progression of the protocol. The raw data is given in Appendix A.

While the main benefit of this scheme is lost if precalculation is executed after the message has been received, in the effort of fair comparison, the whole TSSign protocol is compared to other schemes. It was found that the majority of time is taken with precalculation, as expected, and so the time to run TSSign

from Section 3 was roughly constant even as the threshold increased for the same group size. This is expected because all participants are required in precalculation for any threshold. Choosing the group sizes to be $N = 2t + 1$ for Section 3, the comparison with [14] and [13] are shown in Figure 1.

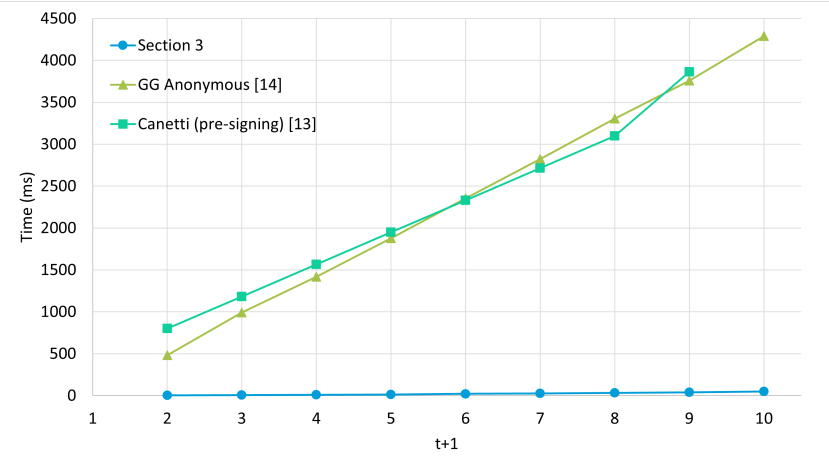

Fig. 1. Comparison of signing timings (including precalculation) of Section 3 with [14] and [13] for t up to 9. [13] includes only precalculation and t up to 8.

All data excludes network latency time for equal comparison with [14] and [13]. Table 1 compares rounds of communication in the three schemes. Since Section 3 has fewer communication rounds, it will be even faster than [14] and [13], when including network latency.

Communication rounds	Section 3	[14]	[13]
Key Generation	2	3	3
Signing protocol	4	6	4

Table 1. Table showing the number of rounds of communication in key generation and signing comparing Section 3 in the presence of eavesdropping adversaries, [14], and [13].

Figure 2 shows the speed of signing after precalculation for the scheme presented in Section 3. That is, Step 9 to 10 in TSSign, taking $N = 2t + 1$ for each t. Given a non-interactive signing time of 0.12ms for $t = 1$, the number of signatures that can be generated per second with this scheme is over 8000.

Finally, the size of communications is compared. The data is given in kB and compared to precalculation in [13] (the data is not available in [14]). The

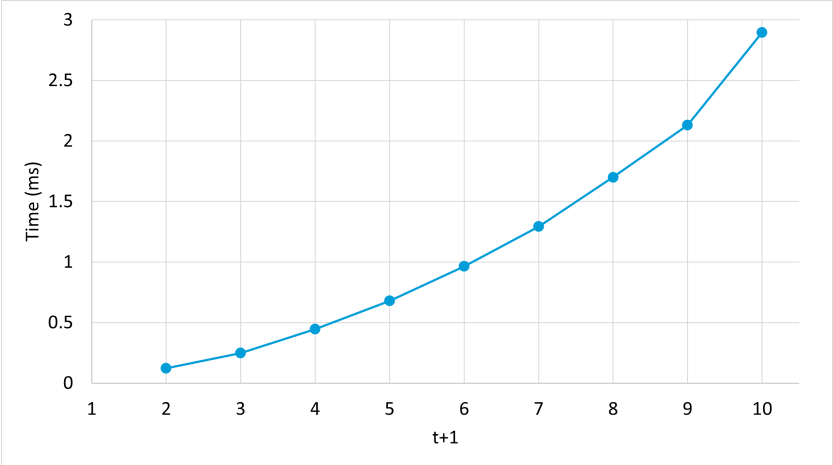

Fig. 2. Time taken to create a signature after precalculation given in milliseconds (ms) for Section 3.

size of communications given for Section 3 includes both precalculation and the non-interactive signing step.

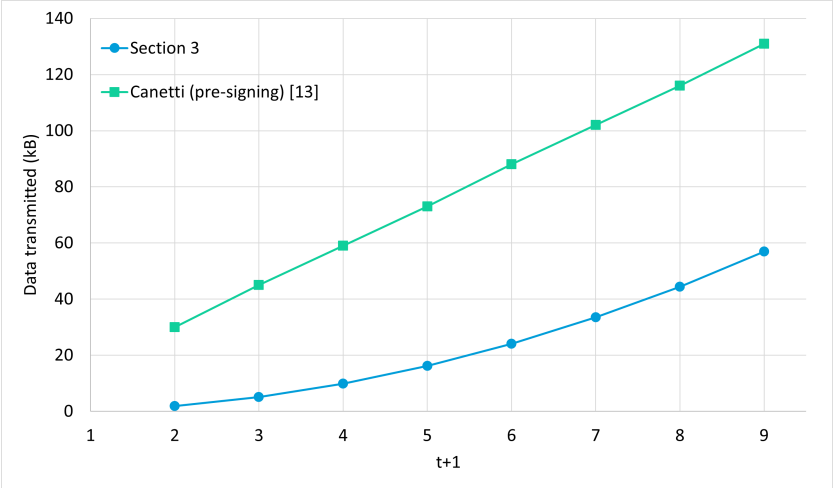

Fig. 3. Size of communication transmitted in kilobytes (kB). The data in [13] is the precalculated data only, and the data for Section 3 includes data for the whole signing protocol.

The communication size in [13] increases linearly in the group size, whilst the data for Section 3 increases quadratically with the group size. This is because all

participants are required during precalculation steps in signing for the scheme in Section 3. This impacts the precalculation stage but has the benefit that any participant can execute the non-interactive phase. In the non-interactive phase, the communication in Section 3 increases linearly with the number of signers.

The scheme in [13] also gives timing and communication size data for part of the key generation algorithm, which has a significant overhead. The data given is for generating Pallier keys (required in both [14] and [13]), which are not required for the scheme in Section 3, and therefore is not comparable. This is additional time and computational complexity in [14] and [13] that is not required in the scheme presented Section 3.

6 Acknowledgements

The author thanks Owen Vaughan, Wei Zhang, Mehmet Sabir Kiraz, and Katharine Molloy for useful comments on the paper. The author also thanks John Murphy and Josie Wilden for implementing the scheme.

References

1. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust threshold dss signatures. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 354–371. Springer (1996)
2. MacKenzie, P., Reiter, M.K.: Two-party generation of dsa signatures. International Journal of Information Security 2(3-4), 218–239 (2004)
3. Lindell, Y.: Fast secure two-party ecdsa signing. In: Annual International Cryptology Conference. pp. 613–644. Springer (2017)
4. Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Secure two-party threshold ecdsa from ecdsa assumptions. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 980–997. IEEE (2018)
5. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Two-party ecdsa from hash proof systems and efficient instantiations. In: Annual International Cryptology Conference. pp. 191–221. Springer (2019)
6. Gennaro, R., Goldfeder, S., Narayanan, A.: Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In: International Conference on Applied Cryptography and Network Security. pp. 156–174. Springer (2016)
7. Gennaro, R., Goldfeder, S.: Fast multiparty threshold ecdsa with fast trustless setup. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1179–1194 (2018)
8. Lindell, Y., Nof, A.: 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. pp. 1837–1854 (2018)
9. Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Threshold ecdsa from ecdsa assumptions: the multiparty case. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 1051–1066. IEEE (2019)
10. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Bandwidth-efficient threshold ec-dsa. In: IACR International Conference on Public-Key Cryptography. pp. 266–296. Springer (2020)

11. Battagliola, M., Longo, R., Meneghetti, A., Sala, M.: Threshold ecdsa with an offline recovery party. arXiv preprint arXiv:2007.04036 (2020)
12. Gagol, A., Straszak, D.: Threshold ecdsa for decentralized asset custody (2020)
13. Canetti, R., Makriyannis, N., Peled, U.: Uc non-interactive, proactive, threshold ecdsa. IACR Cryptol. ePrint Arch. 2020, 492 (2020)
14. Gennaro, R., Goldfeder, S.: One round threshold ecdsa with identifiable abort. IACR Cryptol. ePrint Arch. 2020, 540 (2020)
15. Kravitz, D.W.: Digital signature algorithm (Jul 27 1993), uS Patent 5,231,668
16. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. Journal of Cryptology 20(1), 51–83 (2007)
17. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: 28th Annual Symposium on Foundations of Computer Science (sfcs 1987). pp. 427–438. IEEE (1987)
18. Damgård, I., Jakobsen, T.P., Nielsen, J.B., Pagter, J.I., Østergård, M.B.: Fast threshold ecdsa with honest majority. IACR Cryptol. ePrint Arch. 2020, 501 (2020)
19. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on computing 17(2), 281–308 (1988)

A Raw Data for Benchmarking

The raw data for the graphs shown in Section 5 is given below. The timing for Section 3 is averaged over 20 iterations and the data for [14] and [13] is taken directly from their papers.

$t+1$	TSSign in Section 3	TSSign in [14]	TSSign in [13]
2	3.35	484	801
3	6.12	991	1183
4	9.48	1418	1566
5	13.78	1879	1949
6	21.90	2355	2332
7	28.18	2822	2715
8	33.31	3306	3098
9	40.35	3758	3864
10	48.52	4289	-

Table 2. Timings of the signing protocols in milliseconds, plotted in Figure 1. Section 3 and [14] include both precalculation and the non-interactive steps, whilst [13] is precalculation only.

$t+1$	Time in ms of non-interactive step of TSSign in Section 3
2	0.12
3	0.25
4	0.44
5	0.68
6	0.96
7	1.29
8	1.70
9	2.13
10	2.90

Table 3. Time in milliseconds to execute the non-interactive phase of signing given in Section 3, plotted in Figure 2.

$t+1$	TSSign in Section 3	TSSign in [13]
2	2.01	30
3	5.45	45
4	10.57	59
5	17.38	73
6	25.87	88
7	36.04	102
8	47.90	116
9	61.44	131

Table 4. Communication in kB, plotted in Figure 3. Section 3 includes both precalculation and the non-interactive steps, whilst [13] is precalculation only.