Norwegian University of Science and Technology

TWO-ROUND THRESHOLD LATTICE SIGNATURES FROM THRESHOLD HOMOMORPHIC ENCRYPTION

Kamil Doruk Gur, Jonathan Katz, and **Tjerand Silde**
University of Maryland, College Park and **NTNU**

Contents

Contents

Threshold Cryptography

The goal is that secrets are shared between n parties, and that any threshold $1 \leq t \leq n$ can jointly compute a decryption or signature based on their shares.

This gives security against an adversary corrupting at most $t-1$ parties which cannot complete the computation on its own, and robustness if at least t honest parties are available for the computation to be completed.

Applications

On behalf of a set of people/devices/organizations a threshold can...

- sign transactions and legal documents
- sign authentication challenges or certificates
- decrypt ballots in an electronic voting system
- run pre-processing phases for MPC protocols

Contents

Basic Signature Scheme

The private key is a short $\mathbf{s} \in R_q^{\ell+k}$, and the verification key consists of a matrix $\bar{\mathbf{A}} := [\mathbf{A} \mid \mathbf{I}] \in R_q^{k\times(\ell+k)}$ and vector $\boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s}$. The protocol proceeds as follows:

Basic Signature Scheme

The private key is a short $\mathbf{s} \in R_q^{\ell+k}$, and the verification key consists of a matrix $\bar{\mathbf{A}} := [\mathbf{A} \mid \mathbf{I}] \in R_q^{k \times (\ell+k)}$ and vector $\boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s}$. The protocol proceeds as follows:

1. The prover samples a short-ish vector $\boldsymbol{r} \in R_q^{\ell+k}$ and sends $\boldsymbol{w} := \bar{\mathbf{A}}\boldsymbol{r}$.

Basic Signature Scheme

The private key is a short $\mathbf{s} \in R_q^{\ell+k}$, and the verification key consists of a matrix $\bar{\mathbf{A}} := [\mathbf{A} \mid \mathbf{I}] \in R_q^{k \times (\ell+k)}$ and vector $\boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s}$. The protocol proceeds as follows:

1. The prover samples a short-ish vector $\boldsymbol{r} \in R_q^{\ell+k}$ and sends $\boldsymbol{w} := \bar{\mathbf{A}}\boldsymbol{r}$.
2. The verifier responds with a short challenge $c \in R_q$.

Basic Signature Scheme

The private key is a short $\mathbf{s} \in R_q^{\ell+k}$, and the verification key consists of a matrix $\bar{\mathbf{A}} := [\mathbf{A} \mid \mathbf{I}] \in R_q^{k \times (\ell+k)}$ and vector $\boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s}$. The protocol proceeds as follows:

1. The prover samples a short-ish vector $\boldsymbol{r} \in R_q^{\ell+k}$ and sends $\boldsymbol{w} := \bar{\mathbf{A}}\boldsymbol{r}$.
2. The verifier responds with a short challenge $c \in R_q$.
3. The prover responds with a short vector $\boldsymbol{z} := c \cdot \mathbf{s} + \boldsymbol{r}$.

Basic Signature Scheme

The private key is a short $\mathbf{s} \in R_q^{\ell+k}$, and the verification key consists of a matrix $\bar{\mathbf{A}} := [\mathbf{A} \,|\, \mathbf{I}] \in R_q^{k \times (\ell+k)}$ and vector $\boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s}$. The protocol proceeds as follows:

1. The prover samples a short-ish vector $\boldsymbol{r} \in R_q^{\ell+k}$ and sends $\boldsymbol{w} := \bar{\mathbf{A}}\boldsymbol{r}$.
2. The verifier responds with a short challenge $c \in R_q$.
3. The prover responds with a short vector $\boldsymbol{z} := c \cdot \mathbf{s} + \boldsymbol{r}$.
4. The verifier accepts iff $\boldsymbol{z}$ is short and $\bar{\mathbf{A}}\boldsymbol{z} = c \cdot \boldsymbol{y} + \boldsymbol{w}$.

Basic Signature Scheme

The private key is a short $\mathbf{s} \in R_q^{\ell+k}$, and the verification key consists of a matrix $\bar{\mathbf{A}} := [\mathbf{A} \mid \mathbf{I}] \in R_q^{k\times(\ell+k)}$ and vector $\boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s}$. The protocol proceeds as follows:

1. The prover samples a short-ish vector $\boldsymbol{r} \in R_q^{\ell+k}$ and sends $\boldsymbol{w} := \bar{\mathbf{A}}\boldsymbol{r}$.
2. The verifier responds with a short challenge $c \in R_q$.
3. The prover responds with a short vector $\boldsymbol{z} := c \cdot \mathbf{s} + \boldsymbol{r}$.
4. The verifier accepts iff $\boldsymbol{z}$ is short and $\bar{\mathbf{A}}\boldsymbol{z} = c \cdot \boldsymbol{y} + \boldsymbol{w}$.
5. $\rightarrow$ Non-interactive signature scheme if $c = H(\mathsf{pk}, \boldsymbol{w}, m)$.

Basic n-out-of-n Threshold Scheme

The ith signer holds short vector $\mathbf{s}_i$ where $\mathbf{s} = \sum_{i \in [n]} \mathbf{s}_i$ is the private key. Then, the n signers can run a distributed, two-round signing protocol as follows:

Basic n-out-of-n Threshold Scheme

The ith signer holds short vector $\mathbf{s}_i$ where $\mathbf{s} = \sum_{i \in [n]} \mathbf{s}_i$ is the private key. Then, the n signers can run a distributed, two-round signing protocol as follows:

1. The ith signer chooses a short-ish vector $\boldsymbol{r}_i \in R_q^{\ell+k}$ and sends $\boldsymbol{w}_i := \bar{\mathbf{A}}\boldsymbol{r}_i$.

Basic n-out-of-n Threshold Scheme

The ith signer holds short vector $\mathbf{s}_i$ where $\mathbf{s} = \sum_{i \in [n]} \mathbf{s}_i$ is the private key. Then, the n signers can run a distributed, two-round signing protocol as follows:

1. The ith signer chooses a short-ish vector $\boldsymbol{r}_i \in R_q^{\ell+k}$ and sends $\boldsymbol{w}_i := \bar{\mathbf{A}}\boldsymbol{r}_i$.
2. Each signer computes $\boldsymbol{w} := \sum_{i \in [n]} \boldsymbol{w}_i$ followed by $c := H(\boldsymbol{w})$. The ith signer then sends $\boldsymbol{z}_i := c \cdot \mathbf{s}_i + \boldsymbol{r}_i$.

Basic n-out-of-n Threshold Scheme

The ith signer holds short vector $\mathbf{s}_i$ where $\mathbf{s} = \sum_{i \in [n]} \mathbf{s}_i$ is the private key. Then, the n signers can run a distributed, two-round signing protocol as follows:

1. The ith signer chooses a short-ish vector $\boldsymbol{r}_i \in R_q^{\ell+k}$ and sends $\boldsymbol{w}_i := \bar{\mathbf{A}}\boldsymbol{r}_i$.
2. Each signer computes $\boldsymbol{w} := \sum_{i \in [n]} \boldsymbol{w}_i$ followed by $c := H(\boldsymbol{w})$. The ith signer then sends $\boldsymbol{z}_i := c \cdot \mathbf{s}_i + \boldsymbol{r}_i$.
3. Each signer then computes $\boldsymbol{z} := \sum_{i \in [n]} \boldsymbol{z}_i$ and outputs the signature $(c, \boldsymbol{z})$.

Issues with Secret Sharing

Issues with Secret Sharing

- The shared secret must be short for SIS to be hard

Issues with Secret Sharing

- The shared secret must be short for SIS to be hard
- Individual secrets must be short to allow rejection sampling

Issues with Secret Sharing

- The shared secret must be short for SIS to be hard
- Individual secrets must be short to allow rejection sampling
- The sum of short elements is also short, but...

Issues with Secret Sharing

- The shared secret must be short for SIS to be hard
- Individual secrets must be short to allow rejection sampling
- The sum of short elements is also short, but...
- Secret shared elements are uniformly random

Issues with Random Oracles

Fiat-Shamir signatures require a random oracle to produce challenges, and we cannot evaluate a random oracle using MPC, ZKP, or FHE in a black-box way.

We need a homomorphism to share and combine secrets, but we want to evaluate the random oracle on public input (using communication).

Issues with Zero-Knowledge

Signatures are only (honest-verifier) zero-knowledge when no parties abort.

Then the commit message cannot be sent in the clear if anyone aborts.

But we only learn if anyone aborts after we have computed the challenge...

Contents

BGV Encryption

The BGV encryption scheme consists of the following algorithms:

BGV Encryption

The BGV encryption scheme consists of the following algorithms:

- $\mathsf{KGen}_{\mathsf{BGV}}$: Sample a uniform element $a \in R_q$ along with $s, e \leftarrow D_{\mathsf{KGen}}$, and output the public key $\mathsf{pk} := (a, b) = (a, as + pe)$ and secret key $\mathsf{sk} := s$.

BGV Encryption

The BGV encryption scheme consists of the following algorithms:

- $\mathsf{KGen}_{\mathsf{BGV}}$: Sample a uniform element $a \in R_q$ along with $s, e \leftarrow D_{\mathsf{KGen}}$, and output the public key $\mathsf{pk} := (a, b) = (a, as + pe)$ and secret key $\mathsf{sk} := s$.
- $\mathsf{Enc}_{\mathsf{BGV}}$: On input a public key $\mathsf{pk} = (a, b)$ and a message $m \in R_p$, sample $r, e', e'' \leftarrow D_{\mathsf{Enc}}$ and output the ciphertext $(u, v) = (ar + pe', br + pe'' + m)$.

BGV Encryption

The BGV encryption scheme consists of the following algorithms:

- $\mathsf{KGen}_{\mathsf{BGV}}$: Sample a uniform element $a \in R_q$ along with $s, e \leftarrow D_{\mathsf{KGen}}$, and output the public key $\mathsf{pk} := (a, b) = (a, as + pe)$ and secret key $\mathsf{sk} := s$.
- $\mathsf{Enc}_{\mathsf{BGV}}$: On input a public key $\mathsf{pk} = (a, b)$ and a message $m \in R_p$, sample $r, e', e'' \leftarrow D_{\mathsf{Enc}}$ and output the ciphertext $(u, v) = (ar + pe', br + pe'' + m)$.
- $\mathsf{Dec}_{\mathsf{BGV}}$: On input a secret key $\mathsf{sk} = s$ and a ciphertext (u, v), output the message $m := (v - su \bmod q) \bmod p$.

BGV DistKeyGen

The distributed key generation protocol for BGV works as follows:

BGV DistKeyGen

The distributed key generation protocol for BGV works as follows:

1. $\mathcal{P}_i$ samples s_i and e_i from a distribution D_{KGen}, computes $b_i := as_i + pe_i$.

BGV DistKeyGen

The distributed key generation protocol for BGV works as follows:

1. $\mathcal{P}_i$ samples s_i and e_i from a distribution D_{KGen}, computes $b_i := as_i + pe_i$.
2. $\mathcal{P}_i$ secret shares s_i into $\{s_{i,j}\}_{j\in[n]}$ using t-out-of-n Shamir secret sharing. For each j, $\mathcal{P}_i$ sends $s_{i,j}$ and b_i to party $\mathcal{P}_j$ over a secure channel.

BGV DistKeyGen

The distributed key generation protocol for BGV works as follows:

1. $\mathcal{P}_i$ samples s_i and e_i from a distribution D_{KGen}, computes $b_i := as_i + pe_i$.
2. $\mathcal{P}_i$ secret shares s_i into $\{s_{i,j}\}_{j \in [n]}$ using t-out-of-n Shamir secret sharing. For each j, $\mathcal{P}_i$ sends $s_{i,j}$ and b_i to party $\mathcal{P}_j$ over a secure channel.
3. $\mathcal{P}_i$ computes $b := \sum b_j$, $s'_i = \sum s_{j,i}$, and outputs $\mathsf{pk} = (a, b)$ and $\mathsf{sk}_i = s'_i$.

BGV TDec

The threshold decryption procedure for BGV works as follows:

BGV TDec

The threshold decryption procedure for BGV works as follows:

TDec On input a ciphertext $\mathsf{ctx} = (u, v)$, a decryption key share $\mathsf{sk}_i = s_i$, and a set of users $\mathcal{U}$ of size t, compute $m_i := \lambda_i s u$ using Lagrange coefficient λ_i.

Sample noise $E_i \leftarrow R_q$ s.t $\|E_i\|_\infty \leq 2^{\mathsf{sec}} B_{\mathsf{Dec}}$, then output $d_i := m_i + pE_i$.

BGV TDec

The threshold decryption procedure for BGV works as follows:

TDec On input a ciphertext $\mathsf{ctx} = (u, v)$, a decryption key share $\mathsf{sk}_i = s_i$, and a set of users $\mathcal{U}$ of size t, compute $m_i := \lambda_i s u$ using Lagrange coefficient λ_i.

Sample noise $E_i \leftarrow R_q$ s.t $\|E_i\|_\infty \leq 2^{\mathsf{sec}} B_{\mathsf{Dec}}$, then output $d_i := m_i + pE_i$.

Comb On input a ciphertext $\mathsf{ctx} = (u, v)$ and a set of partial decryption shares $\{d_j\}_{j \in \mathcal{U}}$, it outputs $m := (v - \sum_{j \in \mathcal{U}} d_j) \mod p$.

Contents

Possible Solutions

Possible Solutions

- Use noise drowning techniques to avoid rejection sampling

Possible Solutions

- Use noise drowning techniques to avoid rejection sampling
- Use linearly homomorphic encryption to combine shares

Possible Solutions

- Use noise drowning techniques to avoid rejection sampling
- Use linearly homomorphic encryption to combine shares
- Use t-out-of-n threshold decryption to reconstruct signatures

Main Idea

Keys $\mathbf{s}$ and $(\bar{\mathbf{A}}, \boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s})$ are as before. Instead of sharing $\mathbf{s}$, signers will hold an encryption $\mathsf{ctx}_{\mathbf{s}} = \mathsf{Enc}(\mathbf{s})$ and share the decryption key $\boldsymbol{k}$ in a t-out-of-n fashion:

Main Idea

Keys $\mathbf{s}$ and $(\bar{\mathbf{A}}, \boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s})$ are as before. Instead of sharing $\mathbf{s}$, signers will hold an encryption $\mathsf{ctx}_{\mathbf{s}} = \mathsf{Enc}(\mathbf{s})$ and share the decryption key $\boldsymbol{k}$ in a t-out-of-n fashion:

1. The ith signer chooses a short-ish vector $\boldsymbol{r}_i \in R_q^{\ell+k}$ and sends $\boldsymbol{w}_i := \bar{\mathbf{A}}\boldsymbol{r}_i$. It also sends $\mathsf{ctx}_{\boldsymbol{r}_i}$, an encryption of $\boldsymbol{r}_i$.

Main Idea

Keys $\mathbf{s}$ and $(\bar{\mathbf{A}}, \boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s})$ are as before. Instead of sharing $\mathbf{s}$, signers will hold an encryption $\mathsf{ctx}_{\mathbf{s}} = \mathsf{Enc}(\mathbf{s})$ and share the decryption key $\boldsymbol{k}$ in a t-out-of-n fashion:

1. The ith signer chooses a short-ish vector $\boldsymbol{r}_i \in R_q^{\ell+k}$ and sends $\boldsymbol{w}_i := \bar{\mathbf{A}}\boldsymbol{r}_i$. It also sends $\mathsf{ctx}_{\boldsymbol{r}_i}$, an encryption of $\boldsymbol{r}_i$.
2. Each signer computes $\boldsymbol{w} := \sum_{i \in \mathcal{U}} \boldsymbol{w}_i$, $c = H(\boldsymbol{w})$, and "encrypted signature" $\mathsf{ctx}_{\boldsymbol{z}} := c \cdot \mathsf{ctx}_{\mathbf{s}} + \sum_{i \in \mathcal{U}} \mathsf{ctx}_{\boldsymbol{r}_i}$. It sends its threshold decryption share of $\mathsf{ctx}_{\boldsymbol{z}}$.

Main Idea

Keys $\mathbf{s}$ and $(\bar{\mathbf{A}}, \boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s})$ are as before. Instead of sharing $\mathbf{s}$, signers will hold an encryption $\mathsf{ctx}_{\mathbf{s}} = \mathsf{Enc}(\mathbf{s})$ and share the decryption key $\boldsymbol{k}$ in a t-out-of-n fashion:

1. The ith signer chooses a short-ish vector $\boldsymbol{r}_i \in R_q^{\ell+k}$ and sends $\boldsymbol{w}_i := \bar{\mathbf{A}}\boldsymbol{r}_i$. It also sends $\mathsf{ctx}_{\boldsymbol{r}_i}$, an encryption of $\boldsymbol{r}_i$.

2. Each signer computes $\boldsymbol{w} := \sum_{i \in \mathcal{U}} \boldsymbol{w}_i$, $c = H(\boldsymbol{w})$, and "encrypted signature" $\mathsf{ctx}_{\boldsymbol{z}} := c \cdot \mathsf{ctx}_{\mathbf{s}} + \sum_{i \in \mathcal{U}} \mathsf{ctx}_{\boldsymbol{r}_i}$. It sends its threshold decryption share of $\mathsf{ctx}_{\boldsymbol{z}}$.

3. Given decryption shares from all parties, each signer can decrypt $\mathsf{ctx}_{\boldsymbol{z}}$ to obtain $\boldsymbol{z}$, and output the signature $(c, \boldsymbol{z})$.

Passive Protocol

$\underline{\mathsf{Sign}_{\mathcal{TS}}(\mathsf{sk}_i, \mathsf{aux}, \mathcal{U}, \mu)}$

$r_{i,1}, r_{i,2} \leftarrow D_r, \quad \boldsymbol{r}_i := \begin{bmatrix} r_{i,1} & r_{i,2} \end{bmatrix}$

$w_i := \langle \mathbf{a}, \boldsymbol{r}_i \rangle, \quad \mathsf{ctx}_{\mathbf{r}_i} := \mathsf{Enc}(\mathsf{pk}_{\mathcal{E}}, \boldsymbol{r}_i)$ $\quad \xrightarrow{w_i,\ \mathsf{ctx}_{\mathbf{r}_i}}$

$w := \sum_{j \in \mathcal{U}} w_j, \quad c := H(w, \mathsf{pk}, \mu)$ $\quad \xleftarrow{\{(w_j, \mathsf{ctx}_{\mathbf{r}_j})\}_{j \in \mathcal{U} \setminus \{i\}}}$

$\mathsf{ctx}_{\mathbf{z}} := c \cdot \mathsf{ctx}_{\mathbf{s}} + \sum_{j \in \mathcal{U}} \mathsf{ctx}_{\mathbf{r}_j}$

$\mathsf{ds}_i := \mathsf{TDec}(\mathsf{ctx}_{\mathbf{z}}, \mathsf{sk}_i, \mathcal{U})$ $\quad \xrightarrow{\mathsf{ds}_i}$

$\boldsymbol{z} := \mathsf{Comb}(\mathsf{ctx}_{\mathbf{z}}, \{\mathsf{ds}_j\}_{j \in \mathcal{U}})$ $\quad \xleftarrow{\{\mathsf{ds}_j\}_{j \in \mathcal{U} \setminus \{i\}}}$

$\textbf{return } \sigma := (c, \boldsymbol{z})$

Contents

Possible Solutions

Possible Solutions

- Use linearly homomorphic trapdoor commitments first

Possible Solutions

- Use linearly homomorphic trapdoor commitments first
- Use zero-knowledge proof to ensure correct computation

Possible Solutions

- Use linearly homomorphic trapdoor commitments first
- Use zero-knowledge proof to ensure correct computation
- Use straight-line extractable ZKPs for parallel execution

Actively Secure Signing Protocol

$\underline{\mathsf{Sign}_{\mathcal{TS}}(\mathsf{pk}, \mathsf{sk}_i, \mathsf{aux}, \mathcal{U}, \mu)}$

$\mathsf{ck} := H_2(\mathsf{pk}, \mu), \quad r_{i,1}, r_{i,2} \leftarrow D_r$

$\boldsymbol{r}_i := [r_{i,1} \; r_{i,2}], \quad \rho_i \leftarrow \chi, w_i := \langle \boldsymbol{a}, \boldsymbol{r}_i \rangle$

$\mathsf{com}_i := \mathsf{Com}_{\mathsf{ck}}(w_i, \rho_i), \mathsf{ctx}_{\boldsymbol{r}_i} \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathcal{E}}, \boldsymbol{r}_i)$

Compute $\pi_{\boldsymbol{r}_i}$ according to Section $\boxed{5.1}$ $\quad \xrightarrow{(\mathsf{ctx}_{\boldsymbol{r}_i}, \mathsf{com}_i, \pi_{\boldsymbol{r}_i})}$

$\xleftarrow{\{(\mathsf{ctx}_{\boldsymbol{r}_j}, \mathsf{com}_j, \pi_{\boldsymbol{r}_j})\}_{j \in \mathcal{U} \setminus \{i\}}}$

if any $\pi_{\boldsymbol{r}_j}$ **is invalid**: **abort** (j)

$\mathsf{com} := \sum_{j \in \mathcal{U}} \mathsf{com}_j, \quad c := H_3(\mathsf{com}, \mathsf{pk}, \mu)$

$\mathsf{ctx}_{\boldsymbol{z}} := c \cdot \mathsf{ctx}_{\mathbf{s}} + \sum_{j \in \mathcal{U}} \mathsf{ctx}_{\boldsymbol{r}_j}$

$\mathsf{ds}_i := \mathsf{TDec}(\mathsf{ctx}_{\boldsymbol{z}}, \mathsf{sk}_i, \mathcal{U})$

$\xrightarrow{(w_i, \rho_i, \mathsf{ds}_i)}$

$\xleftarrow{\{(w_j, \rho_j, \mathsf{ds}_j)\}_{j \in \mathcal{U} \setminus \{i\}}}$

if any $\mathsf{Open}(\mathsf{com}_j, w_j, \rho_j) = 0$: **abort** (j)

$\boldsymbol{z} := \mathsf{Comb}(\mathsf{ctx}_{\boldsymbol{z}}, \{\mathsf{ds}_j\}_{j \in \mathcal{U}})$

if $\boldsymbol{z} = \perp$ with **abort** (j): **abort** (j)

$\rho := \sum_{j \in \mathcal{U}} \rho_j$

return $\sigma := (c, \boldsymbol{z}, \rho)$

Contents

Setting

- Signing threshold of $t = 3$ out of $n = 5$ signers
- Signing at most 1 or 365 or 2^{64} signatures total
- Comparing to Dilithium: n keys and t signatures
- Focus on signature and key size, not communication

Performance Estimates

Comm.	σ_1	y_1	Π_1	σ_β	y_β	Π_β
Size	4 KB	3 KB	≈ 750 KB	9 KB	7.5 KB	≈ 750 KB
Comm.	σ_∞	y_∞	Π_∞	σ_{triv}	y_{triv}	Π_{triv}
Size	20 KB	14 KB	≈ 1.5 MB	7.3 KB	6.6 KB	2.4 KB

We present sizes for 3-out-of-5 threshold signatures, where $\beta = 365$ times.

We assume a trusted setup, only allow for sequential execution, and give a rough estimate for communication sizes. An optimistic approach reduces communication by 50 % to the potential cost of 3 rounds of interaction.

Future Work

Future Work

- Use modules instead of rings for a more flexible design (as Dilithium)

Future Work

- Use modules instead of rings for a more flexible design (as Dilithium)
- Instantiate the distributed key generation protocol as well

Future Work

- Use modules instead of rings for a more flexible design (as Dilithium)
- Instantiate the distributed key generation protocol as well
- Detail the communication and optimize parameters and proofs

Future Work

- Use modules instead of rings for a more flexible design (as Dilithium)
- Instantiate the distributed key generation protocol as well
- Detail the communication and optimize parameters and proofs
- Make sure all proofs are online extractable for parallel composition

Future Work

- Use modules instead of rings for a more flexible design (as Dilithium)
- Instantiate the distributed key generation protocol as well
- Detail the communication and optimize parameters and proofs
- Make sure all proofs are online extractable for parallel composition
- Implementing the scheme for more thresholds and signature bounds

Thank you! Questions?

The paper is available at: https://eprint.iacr.org/2023/1318