A Tale of Time Release powered by Blockchain and IBE

Gennaro Avitabile[1] Nico Döttling[2,*] Lucjan Hanzlik[2]
Bernardo Magri[3,4] Christos Sakkas[3] **Stella Wohnig**[2,5]

[1]IMDEA Software Institute

[2]Helmholtz Center for Information Security (CISPA)

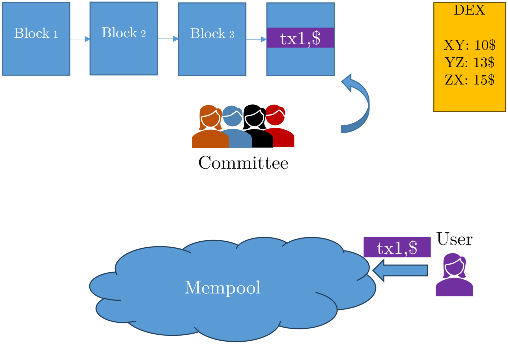
[3]The University of Manchester

[4]Primev [5]Saarland University

[*]Funded by an ERC grant

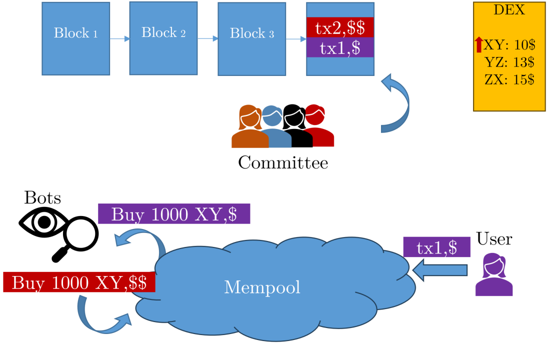
03.05.2025, Madrid

Motivation: Front-Running



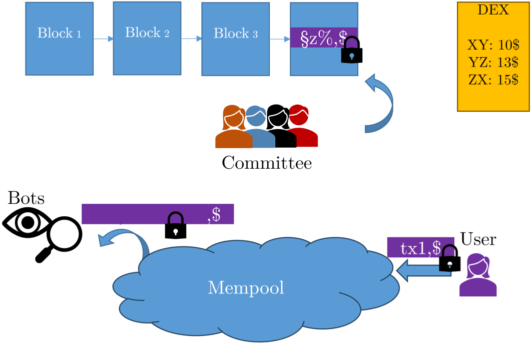
Motivation: Front-Running

is based on re-ordering dependent on content of txs.



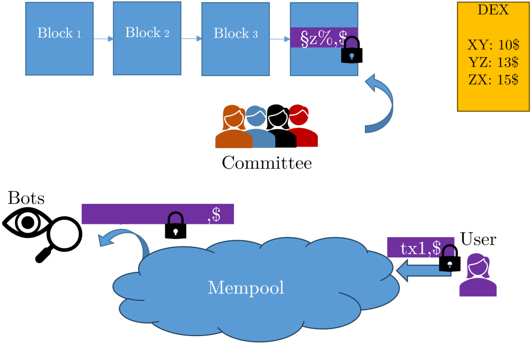
Motivation: Front-Running

is based on re-ordering dependent on content of txs.
So if we hide the content...



Motivation: Front-Running

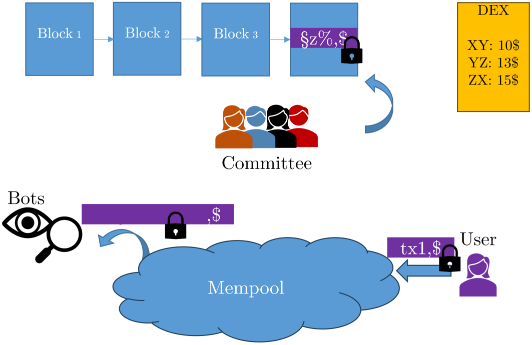
is based on re-ordering dependent on content of txs.
So if we hide the content... JUST long enough



Motivation: Front-Running

is based on re-ordering dependent on content of txs.
So if we hide the content... JUST long enough

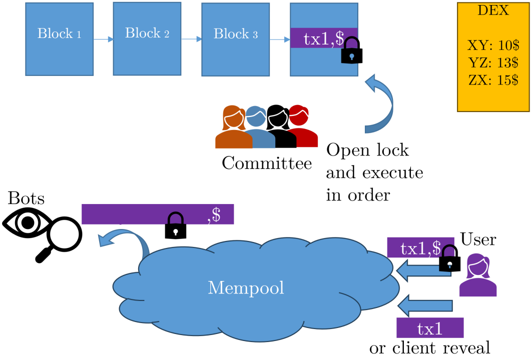
two ways:



Motivation: Front-Running

is based on re-ordering dependent on content of txs.

So if we hide the content... JUST long enough

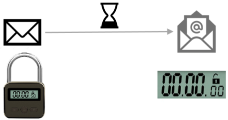


two ways:

Commit & Open

Offline users?

-or-



Time Lock Puzzles

Forced Opening possible!

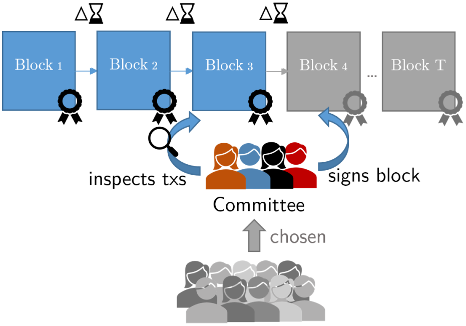
Assumptions to build Time-Lock Puzzles

Assumptions to build Time-Lock Puzzles

Assumptions to build Time-Lock Puzzles

Just use the blockchain as TTP.

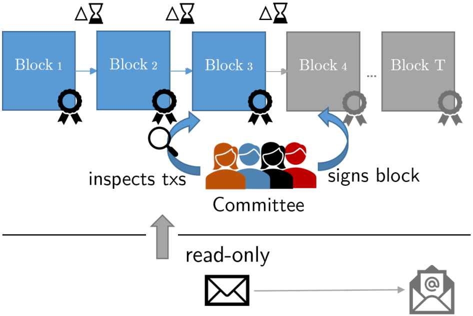
PoS blockchain as TTP?



Observations

- Trust infrastructure
- Periodic signing, Reference clock!
- $\Rightarrow$ Reuse trust, committee work

PoS blockchain as TTP?



Observations

- Trust infrastructure
- Periodic signing, Reference clock!
- $\Rightarrow$ Reuse trust, committee work

Goal: Off-Chain Tool

- Public Enc- & Decryption
- Minimize committee overhead!
- Concrete efficiency

McFly - Verifiable Encryption to the Future Made Practical

Joint work of Nico Döttling[1,*], Lucjan Hanzlik[1], Bernardo Magri[2] and Stella Wohnig[1,3]

Published at Financial Crypto 2023

[1] Helmholtz Center for Information Security (CISPA)

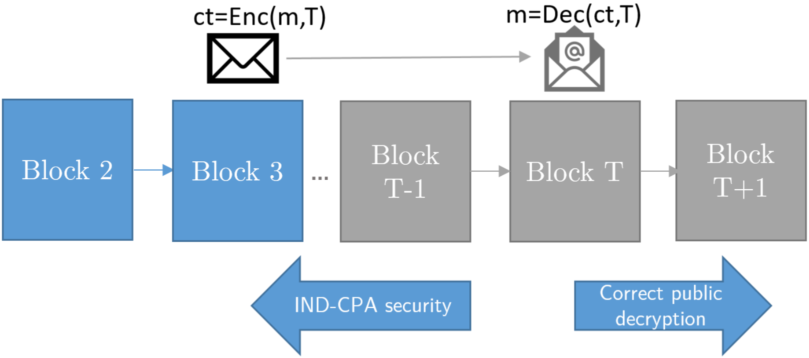
[2] The University of Manchester

[3] Saarland University

* Funded by an ERC grant

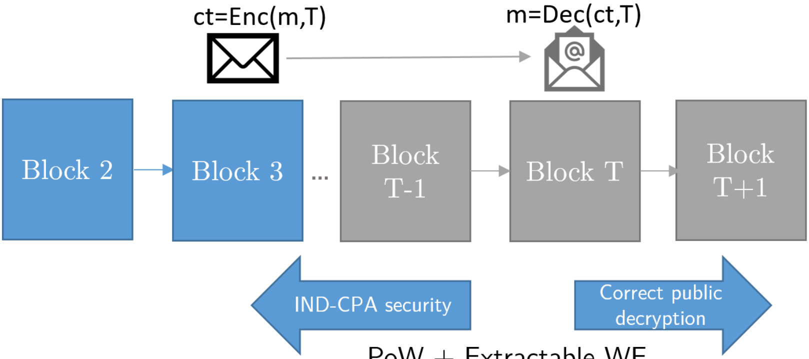
Vision

Time = block height. Users have read-only access to a blockchain.



Previous work [LJKW18] How to build timelock encryption

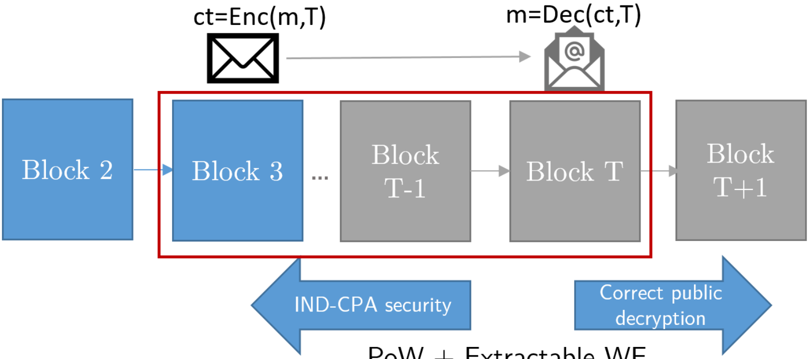
Time = block height. Users have read-only access to a blockchain.



PoW + Extractable WE

Previous work [LJKW18] How to build timelock encryption

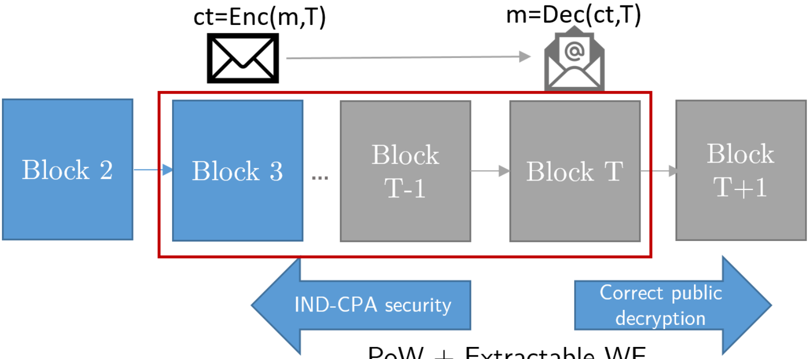
Time = block height. Users have read-only access to a blockchain.



PoW + Extractable WE

Previous work [LJKW18] How to build timelock encryption

Time = block height. Users have read-only access to a blockchain.

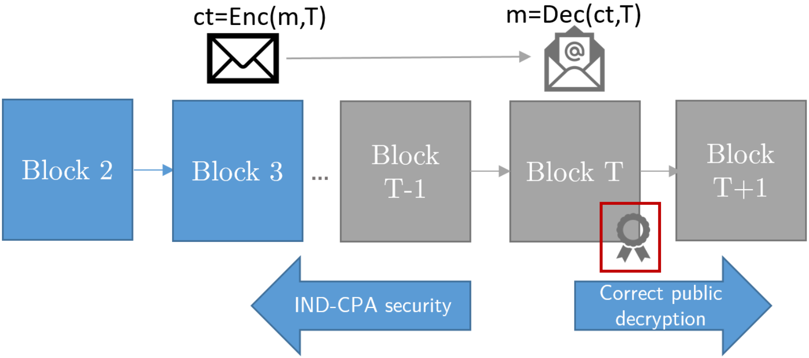


PoW + Extractable WE

Inefficient!

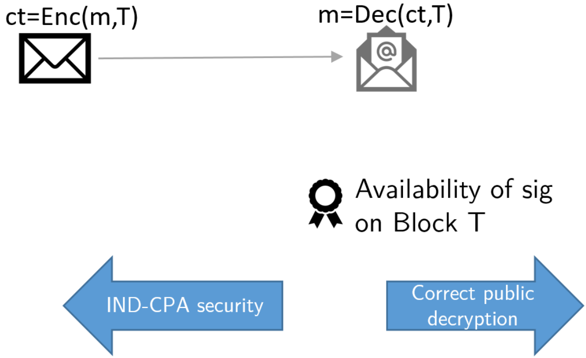
Our idea

Time = block height. Users have read-only access to a blockchain.

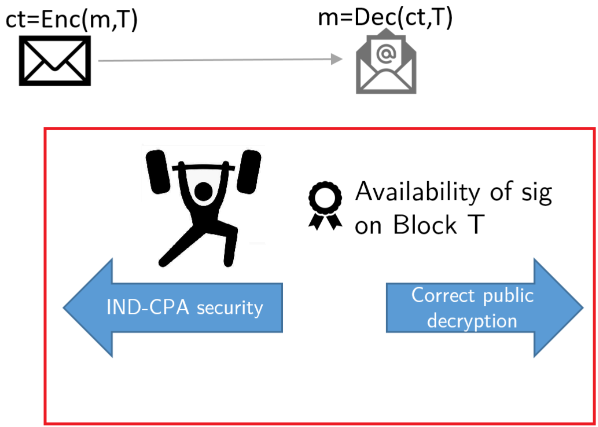


Our idea

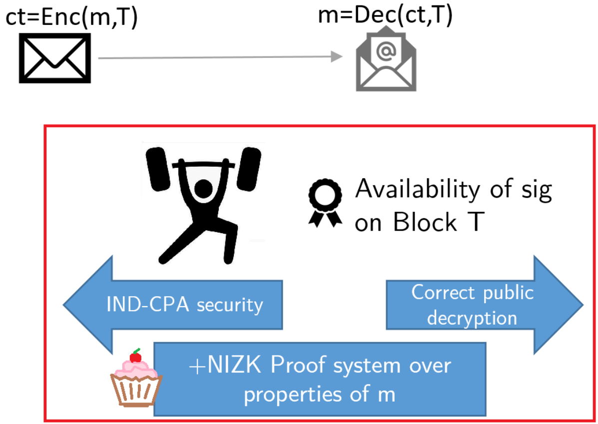
Time = block height. Users have read-only access to a blockchain.



Signature Witness Encryption



Signature Witness Encryption



t-of-n Signature Witness Encryption (basically)

$ct = Enc(m, V, T)$

$m = Dec(ct, V, U, \sigma)$

Encrypt message m

- under potential signer set $V = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$
- and reference message T

s.t. Decryption is possible iff we get indices I signatures $\sigma_i, i \in I$

- s.t. $\forall i \in I : \mathsf{Sig.Verify}(\sigma_i, T, \mathsf{vk}_i) = 1$
- and $|I| \geq t$ (i.e. $\geq t$ parties sign on T)

t-of-n Signature Witness Encryption (basically)

$ct = Enc(m, V, T)$

$m = Dec(ct, V, U, \sigma)$

Encrypt message m

- under potential signer set $V = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$
- and reference message T

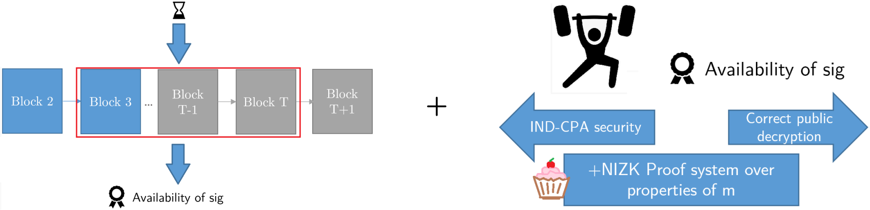
s.t. Decryption is possible iff we get indices I signatures $\sigma_i, i \in I$

- s.t. $\forall i \in I$: $\mathsf{Sig.Verify}(\sigma_i, T, \mathsf{vk}_i) = 1$
- and $|I| \geq t$ (i.e. $\geq t$ parties sign on T)

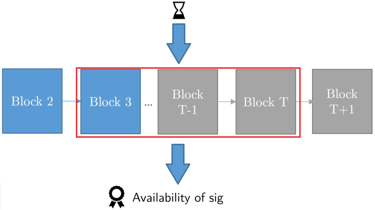
Security: IND-CPA

- For $\leq t - 1$ dishonest keys
- and signing oracle for honest keys (Except for T^*)
- $\mathcal{A}$ chooses m_0, m_1, gets $\mathsf{Enc}(m_b, V, T^*)$, guesses b

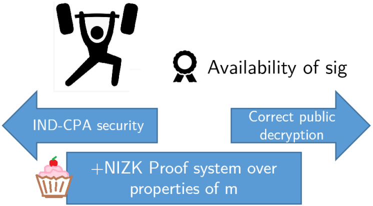
McFly - building on SWE



McFly - building on SWE



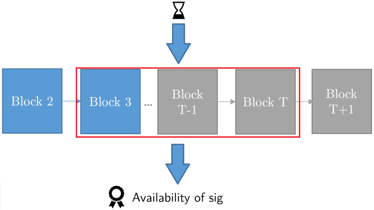
+



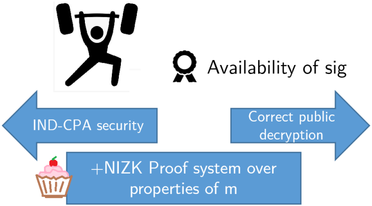
Blockchain assumptions

- Block production rate
- Honest majority
  $\Rightarrow$ No premature signing

McFly - building on SWE



+



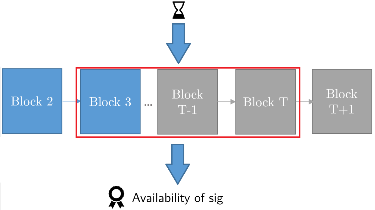
Blockchain assumptions

- Block production rate
- Honest majority
  $\Rightarrow$ No premature signing

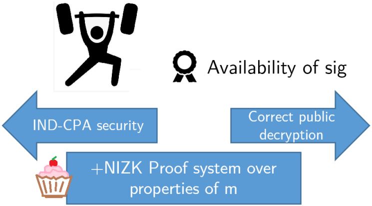
Signature Witness Encryption

- Choose signer set V as committee keys

McFly - building on SWE



+



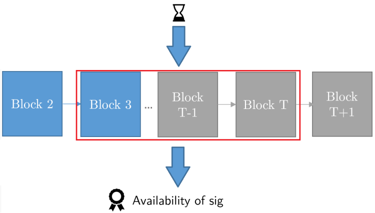
Blockchain assumptions

- Block production rate
- Honest majority
  $\Rightarrow$ No premature signing
- Known committee at decryption time $\Rightarrow$ near-future

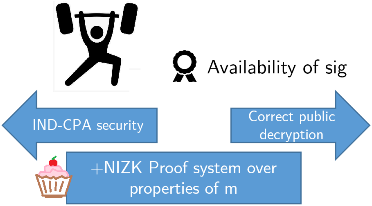
Signature Witness Encryption

- Choose signer set V as committee keys

McFly - building on SWE



+



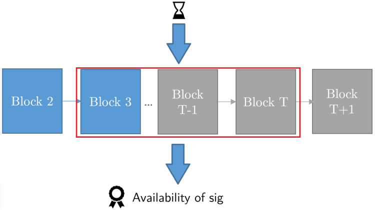
Blockchain assumptions

- Block production rate
- Honest majority
  $\Rightarrow$ No premature signing
- Known committee at decryption time $\Rightarrow$ near-future

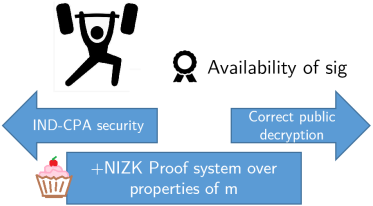
Signature Witness Encryption

- Choose signer set V as committee keys
- Choose reference T as predictable block header of block T.

McFly - building on SWE



+



Blockchain assumptions

- Block production rate
- Honest majority
  $\Rightarrow$ No premature signing
- Known committee at decryption time $\Rightarrow$ near-future
- Custom block structure:
  Sign header seperately!

Signature Witness Encryption

- Choose signer set V as committee keys
- Choose reference T as predictable block header of block T.

SWE plausibility

SWE plausibility

Claim: IBE $\Rightarrow$ SWE

SWE plausibility

Claim: IBE $\Rightarrow$ SWE

- via the Naor transform we know: IBE $\Rightarrow$ Signatures

SWE plausibility

Claim: IBE $\Rightarrow$ SWE

- via the Naor transform we know: IBE $\Rightarrow$ Signatures
- this extends to: IBE $\Rightarrow$ Signatures + SWE

SWE plausibility

Given IBE

- $\mathsf{pk}, \mathsf{msk} \leftarrow \mathsf{Gen}(1^\lambda)$
- $\mathsf{sk}_{ID} \leftarrow \mathsf{Ext}(\mathsf{msk}, ID)$
- $\mathsf{ct} \leftarrow \mathsf{Enc}(m, \mathsf{pk}, ID)$
- $m \leftarrow \mathsf{Dec}(\mathsf{sk}_{ID}, \mathsf{ct}, \mathsf{pk})$

SWE plausibility

Given IBE

- $\mathsf{pk}, \mathsf{msk} \leftarrow \mathsf{Gen}(1^\lambda)$
- $\mathsf{sk}_{ID} \leftarrow \mathsf{Ext}(\mathsf{msk}, ID)$
- $\mathsf{ct} \leftarrow \mathsf{Enc}(m, \mathsf{pk}, ID)$
- $m \leftarrow \mathsf{Dec}(\mathsf{sk}_{ID}, \mathsf{ct}, \mathsf{pk})$

Construct Signature

- Gen: Output IBE.Gen
- $\mathsf{Sign}(M, \mathsf{msk})$: $\sigma_M = \mathsf{sk}_M$
- To verify Enc random message to ID M and see if decryption works

SWE plausibility

Given IBE

- $\mathsf{pk}, \mathsf{msk} \leftarrow \mathsf{Gen}(1^\lambda)$
- $\mathsf{sk}_{ID} \leftarrow \mathsf{Ext}(\mathsf{msk}, ID)$
- $\mathsf{ct} \leftarrow \mathsf{Enc}(m, \mathsf{pk}, ID)$
- $m \leftarrow \mathsf{Dec}(\mathsf{sk}_{ID}, \mathsf{ct}, \mathsf{pk})$

Construct Signature

- Gen: Output IBE.Gen
- $\mathsf{Sign}(M, \mathsf{msk})$: $\sigma_M = \mathsf{sk}_M$
- To verify Enc random message to ID M and see if decryption works

Being able to decrypt is the same as being able to sign!

SWE plausibility

Given IBE

- $\mathsf{pk}, \mathsf{msk} \leftarrow \mathsf{Gen}(1^\lambda)$
- $\mathsf{sk}_{ID} \leftarrow \mathsf{Ext}(\mathsf{msk}, ID)$
- $\mathsf{ct} \leftarrow \mathsf{Enc}(m, \mathsf{pk}, ID)$
- $m \leftarrow \mathsf{Dec}(\mathsf{sk}_{ID}, \mathsf{ct}, \mathsf{pk})$

Construct Signature + 1-of-1 SWE

- Gen: Output IBE.Gen
- Sign(M, msk): $\sigma_M = \mathsf{sk}_M$
- To verify Enc random message to ID M and see if decryption works

Being able to decrypt is the same as being able to sign!

- SWE.Enc($m, V = \{\mathsf{pk}\}, T$): IBE.Enc($m, \mathsf{pk}, ID = T$)
- Decryption with $\sigma_T = \mathsf{sk}_T$ possible, if T is not signed, IND-CPA security from IBE

SWE plausibility

Given IBE

- $\mathsf{pk}, \mathsf{msk} \leftarrow \mathsf{Gen}(1^\lambda)$
- $\mathsf{sk}_{ID} \leftarrow \mathsf{Ext}(\mathsf{msk}, ID)$
- $\mathsf{ct} \leftarrow \mathsf{Enc}(m, \mathsf{pk}, ID)$
- $m \leftarrow \mathsf{Dec}(\mathsf{sk}_{ID}, \mathsf{ct}, \mathsf{pk})$

Construct Signature + 1-of-1 SWE

- Gen: Output IBE.Gen
- $\mathsf{Sign}(M, \mathsf{msk})$: $\sigma_M = \mathsf{sk}_M$
- To verify Enc random message to ID M and see if decryption works

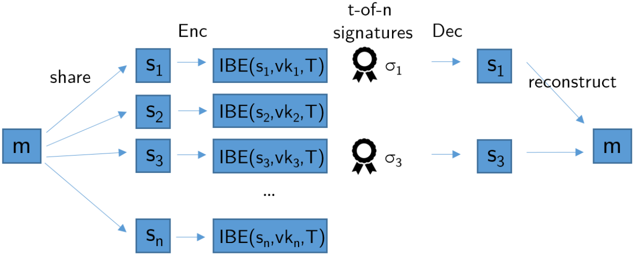
Being able to decrypt is the same as being able to sign!

- $\mathsf{SWE.Enc}(m, V = \{\mathsf{pk}\}, T)$: $\mathsf{IBE.Enc}(m, \mathsf{pk}, ID = T)$
- Decryption with $\sigma_T = \mathsf{sk}_T$ possible, if T is not signed, IND-CPA security from IBE

For t-of-n add secret sharing

Thresholdize

- Assume we know a 1-of-1 SWE
- Thresholdizing it with multiplicative $O(n)$ overhead



Design Choices/Properties of our SWE

- Build for BLS signatures
  - Deployed e.g. in Ethereum
  - Efficient multi-signature

$$\sigma_1 + \sigma_3 = \sigma_{1,3}$$

  - Results from the Boneh-Franklin IBE via the Naor transform
    $\Rightarrow$ 1-of-1 SWE for free

BLS - Quick Recap

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map, with Bilinear Diffie-Hellman assumption in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, and H a hash function.

BLS Signature

Key pairs: $(\mathsf{sk} \in \mathbb{Z}_p, \mathsf{vk} = g_2^{\mathsf{sk}})$

$\mathsf{Sig}(\mathsf{sk}, m) = H(m)^{sk}$

$\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: Output

$e(\sigma, g_2) \stackrel{?}{=} e(H(m), \mathsf{vk})$

BLS - Quick Recap

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map, with Bilinear Diffie-Hellman assumption in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, and H a hash function.

BLS Signature

Key pairs: $(\mathsf{sk} \in \mathbb{Z}_p, \mathsf{vk} = g_2^{\mathsf{sk}})$

$\mathsf{Sig}(\mathsf{sk}, m) = H(m)^{sk}$

$\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: Output

$$e(\sigma, g_2) \stackrel{?}{=} e(H(m), \mathsf{vk})$$

SWE (based on Boneh Franklin)

Encrypt m to vk with reference T:

$$c = g_2^r, c' = (e(H(T), \mathsf{vk}))^r \cdot g_T^m$$

Notice $(e(H(T), \mathsf{vk}))^r = e(\sigma, c)$

BLS - Quick Recap

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map, with Bilinear Diffie-Hellman assumption in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, and H a hash function.

BLS Signature

Key pairs: $(\mathsf{sk} \in \mathbb{Z}_p, \mathsf{vk} = g_2^{\mathsf{sk}})$

$\mathsf{Sig}(\mathsf{sk}, m) = H(m)^{sk}$

$\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: Output

$e(\sigma, g_2) \stackrel{?}{=} e(H(m), \mathsf{vk})$

SWE (based on Boneh Franklin)

Encrypt m to vk with reference T:

$c = g_2^r, c' = (e(H(T), \mathsf{vk}))^r \cdot g_T^m$

Notice $(e(H(T), \mathsf{vk}))^r = e(\sigma, c)$

Multi Signature

$Agg(\sigma_1, \ldots, \sigma_n) = \prod_{i \in [n]} \sigma_i$

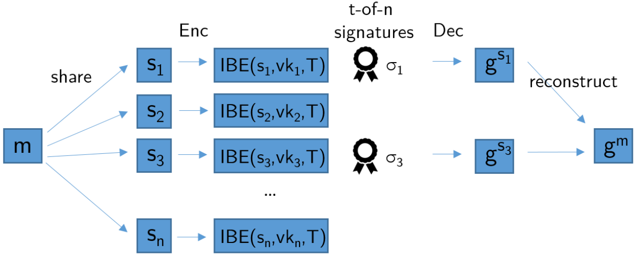
AggVerify for m and $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$:

$e(\sigma, g) \stackrel{?}{=} \prod_{i \in [n]} e(H(m), \mathsf{vk}_i)$

Multi-Signature support? - Remember thresholdizing

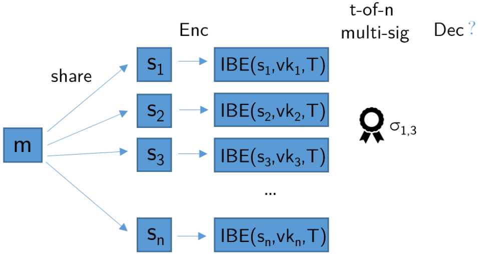
Shamir's secret sharing: Split m into n shares $s_1, \ldots, s_n$ s.t. for any $\geq t$ shares we get $m = \sum_j s_j L_j$, for L_j Lagrange coefficients

No information from $< t$ shares



Multi-Signature support? - Remember thresholdizing

Shamir's secret sharing: Split m into n shares $s_1, \ldots, s_n$
s.t. for any $\geq t$ shares we get $m = \sum_j s_j L_j$, for L_j Lagrange coefficients
No information from $< t$ shares

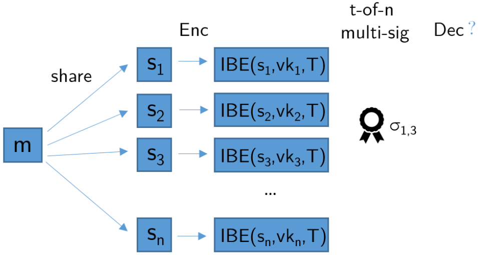


What about multi-signatures?
Have: $c_i' = g^{s_i} \cdot e(\sigma_i, {g_2}^r)$, aggregated $\sigma_I = \Pi_{i \in I} \sigma_i$

Multi-Signature support? - Remember thresholdizing

Shamir's secret sharing: Split m into n shares $s_1, \ldots, s_n$ s.t. for any $\geq t$ shares we get $m = \sum_j s_j L_j$, for L_j Lagrange coefficients

No information from $< t$ shares



What about multi-signatures?

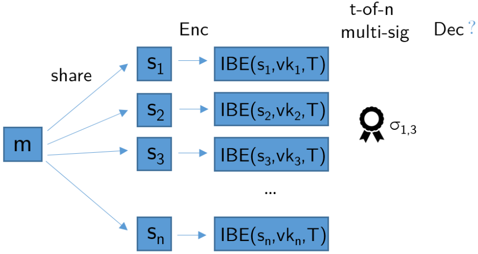
Have: $c_i' = g^{s_i} \cdot e(\sigma_i, g_2{}^r)$, aggregated $\sigma_I = \prod_{i \in I} \sigma_i$

Want: $g^m = \prod_{i \in I} g^{s_i L_i} = \prod_{i \in I} (c_i'^{L_i} / e(\sigma_i, g_2{}^r)^{L_i})$

Multi-Signature support? - Remember thresholdizing

Shamir's secret sharing: Split m into n shares $s_1, \ldots, s_n$ s.t. for any $\geq t$ shares we get $m = \sum_j s_j L_j$, for L_j Lagrange coefficients

No information from $< t$ shares



What about multi-signatures?

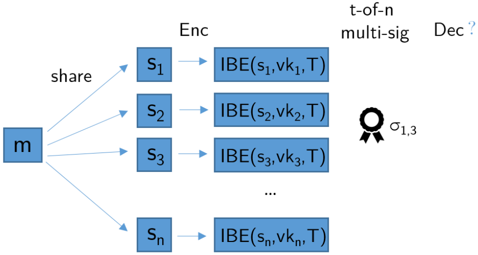
Have: $c_i' = g^{s_i} \cdot e(\sigma_i, {g_2}^r)$, aggregated $\sigma_I = \Pi_{i \in I} \sigma_i$

Want: $g^m = \Pi_{i \in I} g^{s_i L_i} = \Pi_{i \in I} ({c_i'}^{L_i} / e(\sigma_i, {g_2}^r)^{L_i})$

Modified Aggregation: $\sigma_I = \Pi_{i \in I} \sigma_i^{L_i}$

Multi-Signature support? - Remember thresholdizing

Shamir's secret sharing: Split m into n shares $s_1, \ldots, s_n$
s.t. for any $\geq t$ shares we get $m = \sum_j s_j L_j$, for L_j Lagrange coefficients
No information from $< t$ shares



What about multi-signatures?
Have: $c_i' = g^{s_i} \cdot e(\sigma_i, {g_2}^r)$, aggregated $\sigma_I = \Pi_{i \in I} \sigma_i$
Want: $g^m = \Pi_{i \in I} {g^{s_i}}^{L_i} = \Pi_{i \in I} ({c_i'}^{L_i} / e(\sigma_i, {g_2}^r)^{L_i})$
Modified Aggregation: $\sigma_I = \Pi_{i \in I} \sigma_i^{L_i}$ OR workaround as a service

Is that the time?!?!

Design Choices/Properties of our SWE

- Build for BLS signatures
- Supports Multi-Signatures (at a price)

Design Choices/Properties of our SWE

- Build for BLS signatures
- Supports Multi-Signatures (at a price)
- Our SWE is a homomorphic commitment
  - supports commit & open, optimistic settings

Design Choices/Properties of our SWE

- Build for BLS signatures
- Supports Multi-Signatures (at a price)
- Our SWE is a homomorphic commitment
  - supports commit & open, optimistic settings
- Our SWE is verifiable
  - integration with bulletproofs for efficiently proving properties of the contained message
  - this is done by adding a Pederson commitment and a proof of plaintext equality

Design Choices/Properties of our SWE

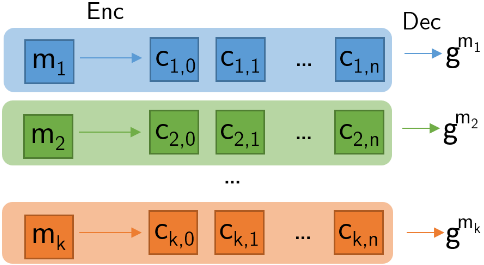
- Build for BLS signatures
- Supports Multi-Signatures (at a price)
- Our SWE is a homomorphic commitment
  - supports commit & open, optimistic settings
- Our SWE is verifiable
  - integration with bulletproofs for efficiently proving properties of the contained message
  - this is done by adding a Pederson commitment and a proof of plaintext equality
- Optimizes for concrete efficiency

Efficiency?

Let's encrypt k messages, with n committee members:

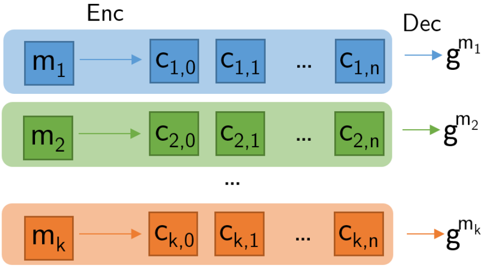
Efficiency?

Let's encrypt k messages, with n committee members:



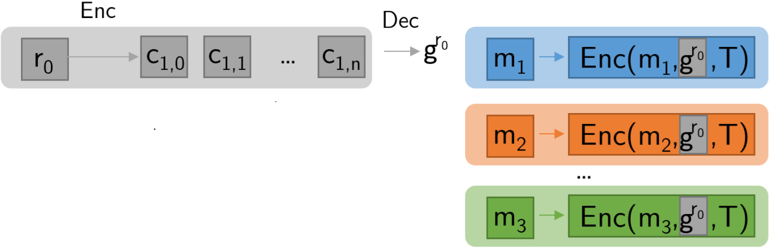
Efficiency?

Let's encrypt k messages, with n committee members: Size $O(n \cdot k)$



Efficiency?

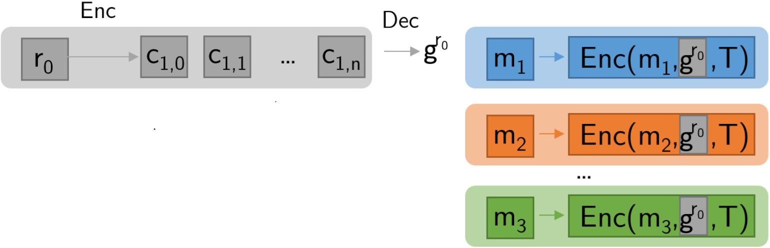
Let's encrypt k messages, with n committee members: Size $O(n \cdot k)$



- Batching: Size $O(n + k)$

Efficiency?

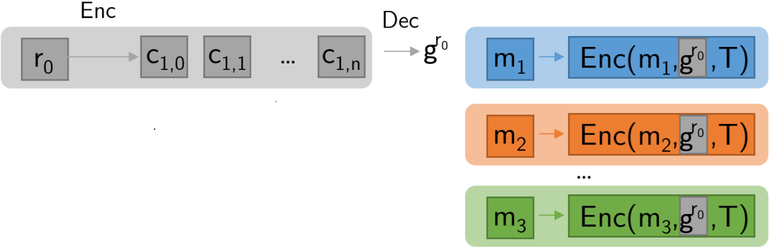
Let's encrypt k messages, with n committee members: Size $O(n \cdot k)$



- Batching: Size $O(n + k)$
- Packing: Allow $m \in \{0, \ldots, 2^\ell - 1\}$ instead of bit messages (baby-step-giant-step)

Efficiency?

Let's encrypt k messages, with n committee members: Size $O(n \cdot k)$



- Batching: Size $O(n + k)$
- Packing: Allow $m \in \{0, \ldots, 2^\ell - 1\}$ instead of bit messages (baby-step-giant-step)
- Bilinear setup: $e : G_1 \times G_2 \to G_T$. Group operations in G_T usually most expensive. We move most operations into G_2!

Now what?

We built concretely efficient verifiable time release encryption

- from BDH and blockchain trust-assumptions
- with small overhead for committee
  (2 signatures per block + changed aggregation)
- for a limited time horizon. $\rightarrow$ Future work!

Now what?

We built concretely efficient verifiable time release encryption

- from BDH and blockchain trust-assumptions
- with small overhead for committee
  (2 signatures per block + changed aggregation)
- for a limited time horizon. $\rightarrow$ Future work!
- [$FMMMTV'$22]: Cryptographic Oracle-based Conditional Payments
  - Similar primitive (VweTS)
  - Cool application: Construct blockchain payments conditioned on real-life events

  Applications of SWE potentially see growing signer sets!
- Asymptotics of O(n) in number of potential signers not ideal
  $\rightarrow$ Future work!

tlock by [Gailly, Melissaris, Romailler]

tlock by [Gailly, Melissaris, Romailler]

- Similar IBE based construction for BLS
- Found perfect deployment environment: drand
  relatively fixed committee regularly signs predictable messages

tlock by [Gailly, Melissaris, Romailler]

- Similar IBE based construction for BLS
- Found perfect deployment environment: drand
  relatively fixed committee regularly signs predictable messages
- At Setup, a distributed key generation was run to get a long-term committee public verification key and a signing key share for each participant such that a threshold number of participants can sign under the public key.

tlock by [Gailly, Melissaris, Romailler]

- Similar IBE based construction for BLS
- Found perfect deployment environment: drand
  relatively fixed committee regularly signs predictable messages
- At Setup, a distributed key generation was run to get a long-term committee public verification key and a signing key share for each participant such that a threshold number of participants can sign under the public key.
- In this setting: 1-of-1 SWE = BF IBE is enough!
- NO committe overhead, long-term encryption, O(1) ciphertext
- Runs in production, with a cute GUI webapp

tlock by [Gailly, Melissaris, Romailler]

- Similar IBE based construction for BLS
- Found perfect deployment environment: drand
  relatively fixed committee regularly signs predictable messages
- At Setup, a distributed key generation was run to get a long-term committee public verification key and a signing key share for each participant such that a threshold number of participants can sign under the public key.
- In this setting: 1-of-1 SWE = BF IBE is enough!
- NO committe overhead, long-term encryption, O(1) ciphertext
- Runs in production, with a cute GUI webapp
- Highly permissioned system

Now what?

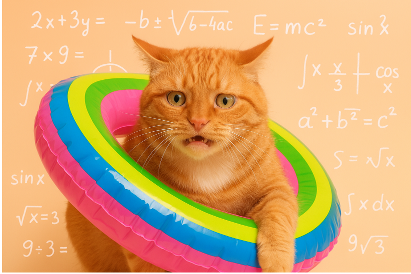
Maybe this is enough for your application and you can tune out now :)

Now what?

Maybe this is enough for your application and you can tune out now :)

Now what?

Maybe this is enough for your application and you can tune out now :)



But a theorist might want:

- Long-Term Encryption across Committees
- Better asymptotics without joint key setup

Long-Term encryption across committees?

Solved if we allow the committee to be more involved

Solutions with near-future solution + bootstrapping to far-future by auxiliary committees

Long-Term encryption across committees?

Solved if we allow the committee to be more involved

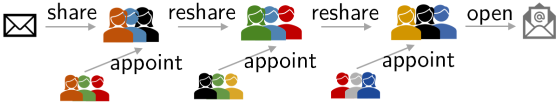
Solutions with near-future solution + bootstrapping to far-future by auxiliary committees

Simple example: [BGGHKLRR20] - Can a public blockchain keep a secret?

Long-Term encryption across committees?

Solved if we allow the committee to be more involved
Solutions with near-future solution + bootstrapping to far-future by auxiliary committees

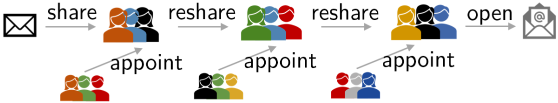
Simple example: [BGGHKLRR20] - Can a public blockchain keep a secret?



Long-Term encryption across committees?

Solved if we allow the committee to be more involved
Solutions with near-future solution + bootstrapping to far-future by auxiliary committees

Simple example: [BGGHKLRR20] - Can a public blockchain keep a secret?



To my knowledge no "off-chain" solution known to extend a time release among multiple committees

$\Rightarrow$ YOUR future work?

Signature-based Witness Encryption with Compact Ciphertext

Joint work of Gennaro Avitabile[1], Nico Döttling[2,*], Bernardo Magri[3], Christos Sakkas[3], Stella Wohnig[2,4]

Published at Asiacrypt 2024

[1] IMDEA Software Institute

[2] CISPA Helmholtz Center for Information Security

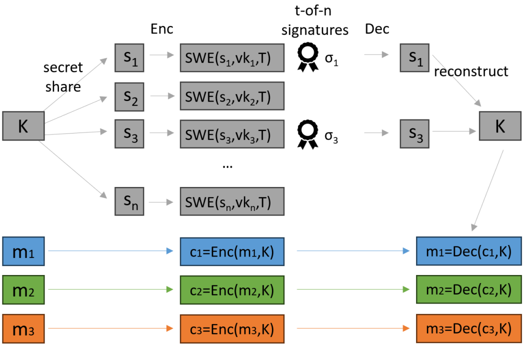
[3] The University of Manchester

[4] Saarland University

* Funded by an ERC grant

High-level Overview so far

- Assume we know a 1-of-1 SWE (BF-IBE in BLS case)
- Thresholdizing it with multiplicative $O(n)$ overhead



Batching via Symmetric Encryption

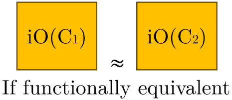
- How CAN we do better?

iO to the rescue

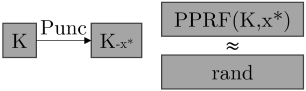
iO to the rescue

The superpower:

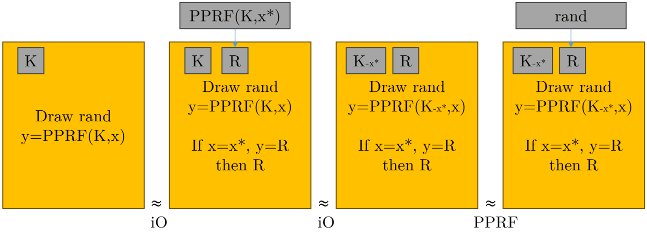
replace circuits



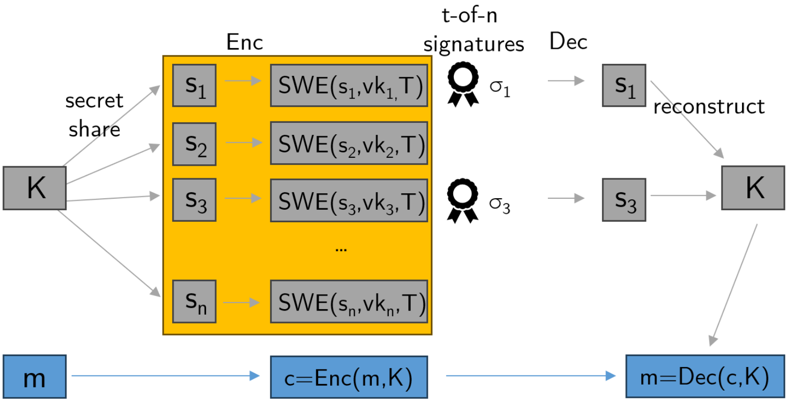
puncture a key to forget a value



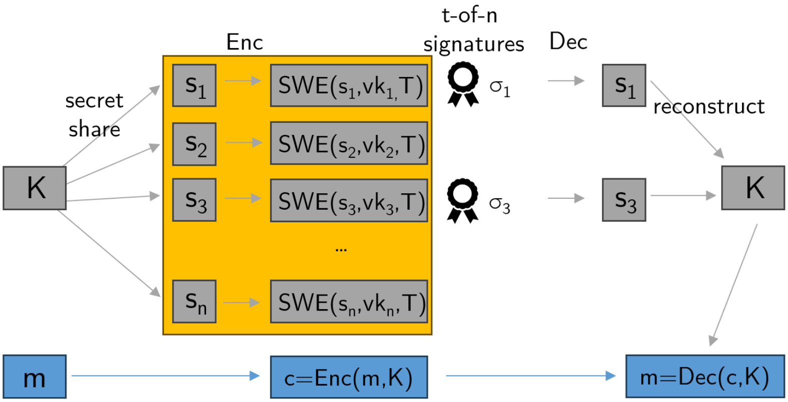
Together they help *forgetting a value* in a circuit!



High-level idea

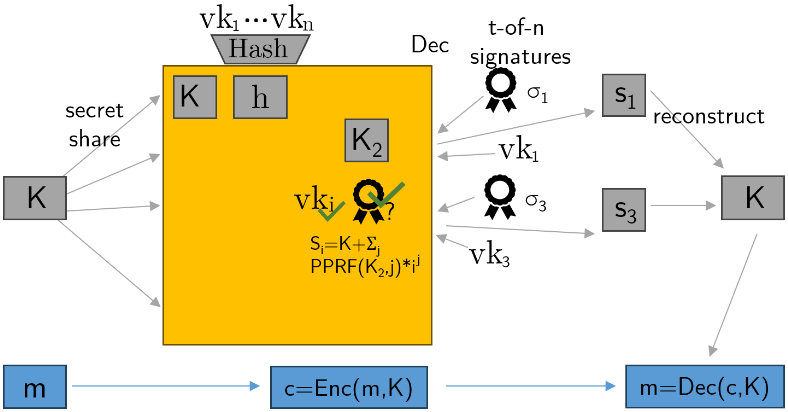


High-level idea



Let a circuit create the shares.

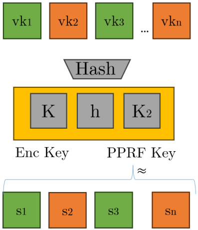
High-level idea



Let a circuit create the shares.
Circuit gets $(i, \mathrm{vk}_i, \sigma_i)$, if valid returns share s_i (pulled from PPRF)
This is O(polylog n).

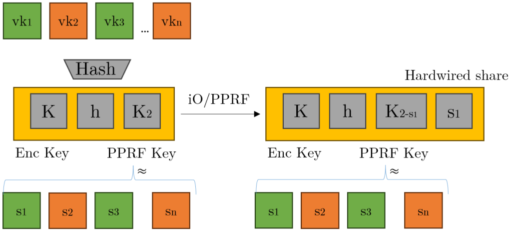
Security Sketch

Fully non-adaptive: reference T, corrupted parties known, all keys honestly created
Want: Forget all honest shares, use secret sharing security.
Intuition: Honest signatures are never given to $\mathcal{A}$ so the circuit is never queried successfully to output honest share.



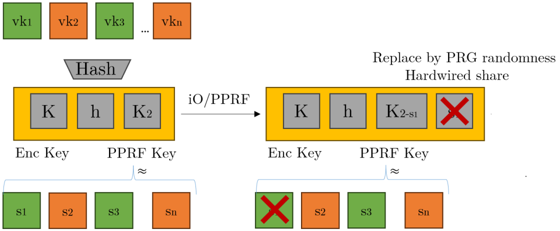
Security Sketch

Fully non-adaptive: reference T, corrupted parties known, all keys honestly created
Want: Forget all honest shares, use secret sharing security.
Intuition: Honest signatures are never given to $\mathcal{A}$ so the circuit is never queried successfully to output honest share.



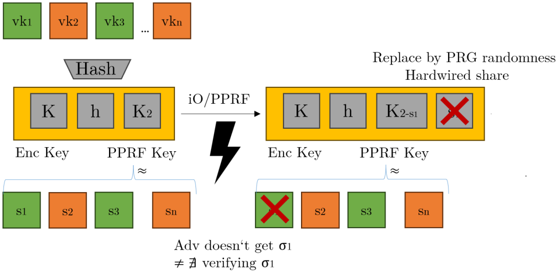
Security Sketch

Fully non-adaptive: reference T, corrupted parties known, all keys honestly created
Want: Forget all honest shares, use secret sharing security.
Intuition: Honest signatures are never given to $\mathcal{A}$ so the circuit is never queried successfully to output honest share.



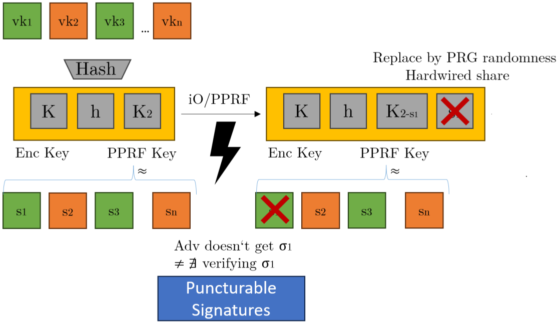
Security Sketch

Fully non-adaptive: reference T, corrupted parties known, all keys honestly created
Want: Forget all honest shares, use secret sharing security.
Intuition: Honest signatures are never given to $\mathcal{A}$ so the circuit is never queried successfully to output honest share.



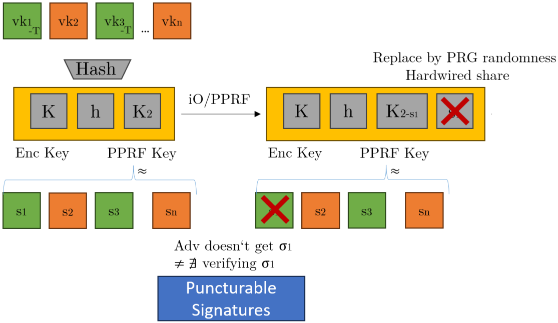
Security Sketch

Fully non-adaptive: reference T, corrupted parties known, all keys honestly created
Want: Forget all honest shares, use secret sharing security.
Intuition: Honest signatures are never given to $\mathcal{A}$ so the circuit is never queried successfully to output honest share.



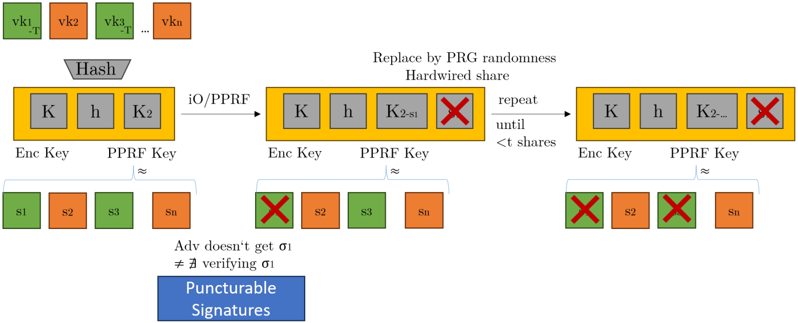
Security Sketch

Fully non-adaptive: reference T, corrupted parties known, all keys honestly created
Want: Forget all honest shares, use secret sharing security.
Intuition: Honest signatures are never given to $\mathcal{A}$ so the circuit is never queried successfully to output honest share.
Puncture all honest keys



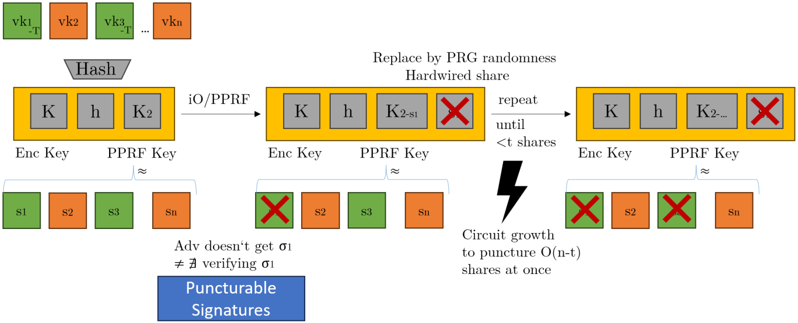
Security Sketch

Fully non-adaptive: reference T, corrupted parties known, all keys honestly created
Want: Forget all honest shares, use secret sharing security.
Intuition: Honest signatures are never given to $\mathcal{A}$ so the circuit is never queried successfully to output honest share.
Puncture all honest keys



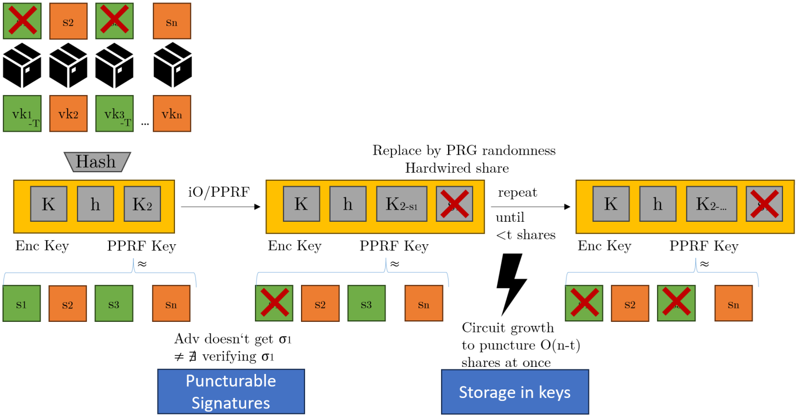
Security Sketch

Fully non-adaptive: reference T, corrupted parties known, all keys honestly created
Want: Forget all honest shares, use secret sharing security.
Intuition: Honest signatures are never given to $\mathcal{A}$ so the circuit is never queried successfully to output honest share.
Puncture all honest keys



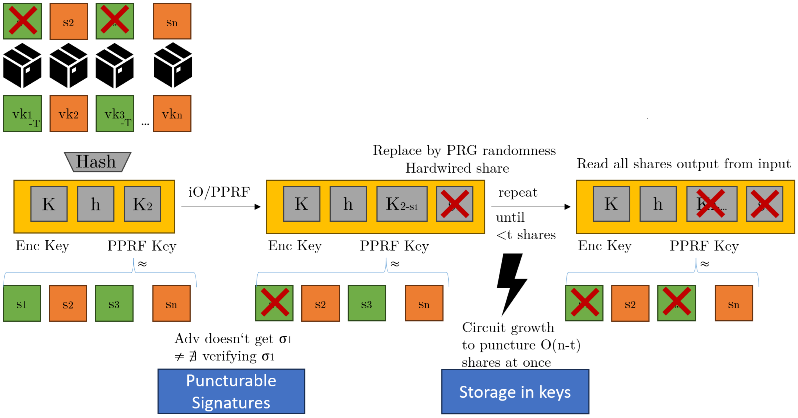
Security Sketch

Fully non-adaptive: reference T, corrupted parties known, all keys honestly created
Want: Forget all honest shares, use secret sharing security.
Intuition: Honest signatures are never given to $\mathcal{A}$ so the circuit is never queried successfully to output honest share.
Puncture all honest keys

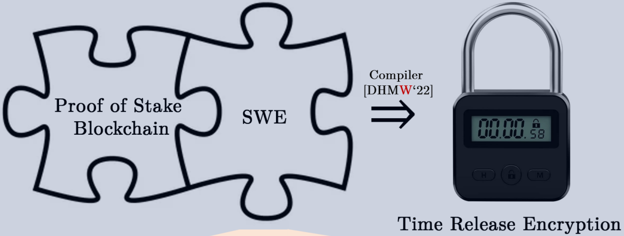


Security Sketch

Fully non-adaptive: reference T, corrupted parties known, all keys honestly created
Want: Forget all honest shares, use secret sharing security.
Intuition: Honest signatures are never given to $\mathcal{A}$ so the circuit is never queried successfully to output honest share.
Puncture all honest keys



Comparison



Concrete efficiency?

- BLS based construction
- Size O(n)
- Based on BDH, ROM

SWE, [DHMW'22]

Asymptotic efficiency?

- Generic construction
- Size O(polylog n)
- Uses iO

cSWE, [ADMSW'24]

Future work

- Are there less heavy compact constructions?
- Find a bootstrapping to the far-future for McFly with no committee communication/joint key setup?
- Any chance if reference messages are not (fully) known?