A Threshold Signature Protocol Implementation via JavaCard API

BBS 2023

CHESS

Antonín Dufka
dufkan@mail.muni.cz
Masaryk University, Brno, Czech Republic

CROCS
Centre for Research on Cryptography and Security

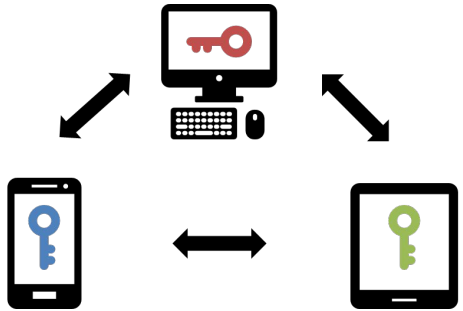
Outline

- Introduction
- Threshold signatures
  - Secure secret sharing
  - Threshold protocols
- Problems in use of threshold signatures
- JCFROST implementation
  - FROST protocol
  - JavaCard smartcards
  - Performance measurements
  - Demonstration
- Conclusion

Threshold signatures

- A concept mainly developed 90’s is now starting to be used in modern applications
  - Cryptocurrencies (multi-signature wallets)
  - Authentication (eIDAS-compliant – SplitKey/SmartID)

Single party computation

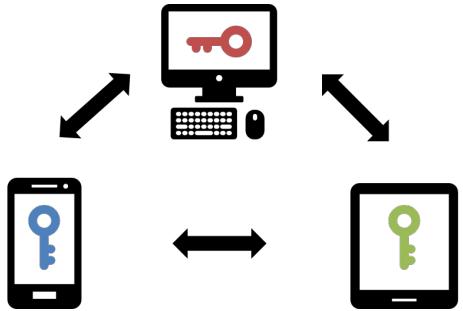


Threshold computation

Threshold signatures

- A concept mainly developed 90's is now starting to be used in modern applications
  - Cryptocurrencies (multi-signature wallets)
  - Authentication (eIDAS-compliant – SplitKey/SmartID)
- A convenient private key protection method
  - Decentralization of storage
  - Elimination of single point of failure
  - Easy integration with existing systems (backward compatibility)

Single party computation

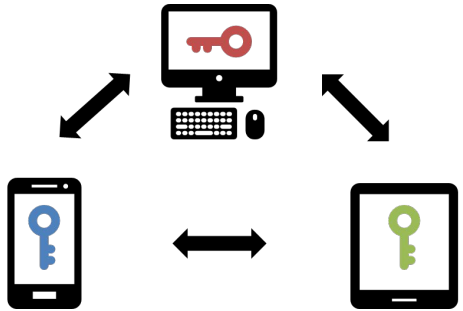


Threshold computation

Secure secret sharing

- Splitting a secret into **n** shares
  - At least **t** shares are needed to reconstruct the secret
  - Less than **t** shares gives *no information* about the secret
  - Can be created “bottom-up” using distributed key generation

Single party computation



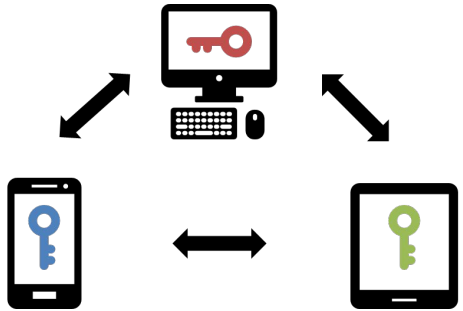
Threshold computation

Secure secret sharing

- Splitting a secret into **n** shares
  - At least **t** shares are needed to reconstruct the secret
  - Less than **t** shares gives *no information* about the secret
  - Can be created “bottom-up” using distributed key generation

- Additive secret sharing (**n**-of-**n**)
  - All shares are needed
  - More efficient to use with protocols
  - Supports non-interactive key generation

Single party computation

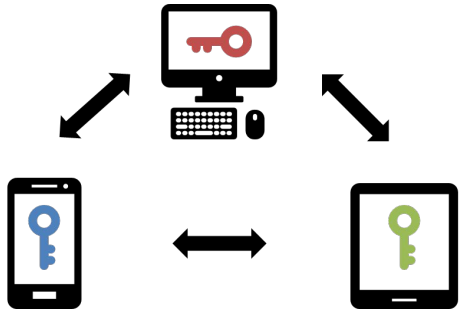


Threshold computation

Secure secret sharing

- Splitting a secret into **n** shares
  - At least **t** shares are needed to reconstruct the secret
  - Less than **t** shares gives *no information* about the secret
  - Can be created “bottom-up” using distributed key generation
- Additive secret sharing (**n**-of-**n**)
  - All shares are needed
  - More efficient to use with protocols
  - Supports non-interactive key generation
- Shamir’s secret sharing (**t**-of-**n**)
  - Only **t** shares are needed to reconstruct the secret

Single party computation

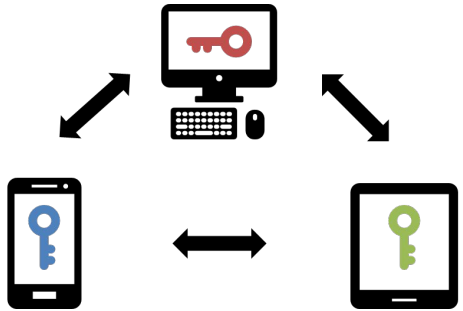


Threshold computation

Threshold protocols

- Devices operate with secret shares directly
  - No need to reconstruct the secret

Single party computation

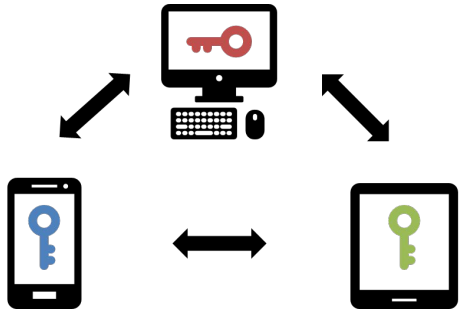


Threshold computation

Threshold protocols

- Devices operate with secret shares directly
  - No need to reconstruct the secret
- Output is indistinguishable from single-party output

Single party computation



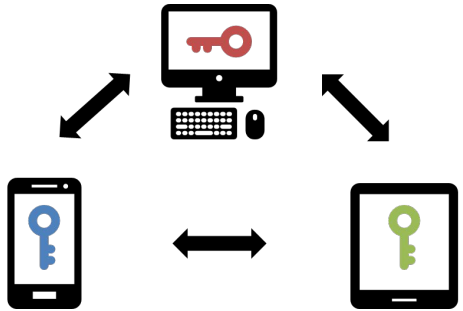
Threshold computation

Threshold protocols

- Devices operate with secret shares directly
  - No need to reconstruct the secret
- Output is indistinguishable from single-party output

- Specialized protocol for each signature type
  - RSA, ECDSA, EdDSA, …

Single party computation



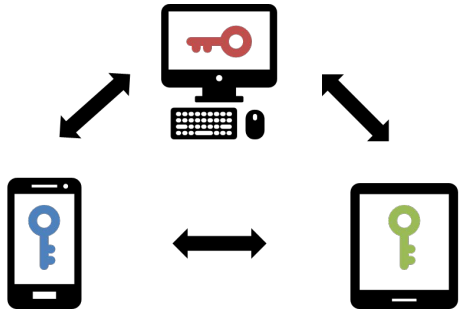
Threshold computation

Threshold protocols

- Devices operate with secret shares directly
  - No need to reconstruct the secret
- Output is indistinguishable from single-party output

Single party computation

- Specialized protocol for each signature type
  - RSA, ECDSA, EdDSA, …
- Typically require
  - Multiple communication rounds
  - More complex computation compared to the base operation

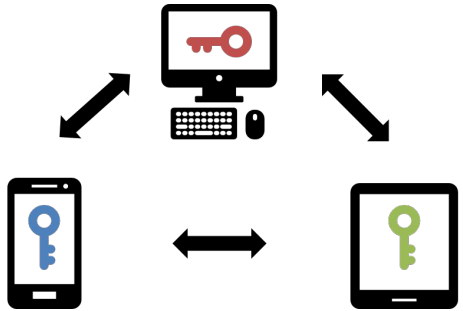


Threshold computation

Practical problems of threshold protocols

- Protocol selection
  - Several protocols exist for each signature signature type
  - Computation (operations, complexity, …)
  - Communication (rounds, bandwidth, reliable broadcast, …)
  - Security assumptions (standard vs less conventional)

Single party computation

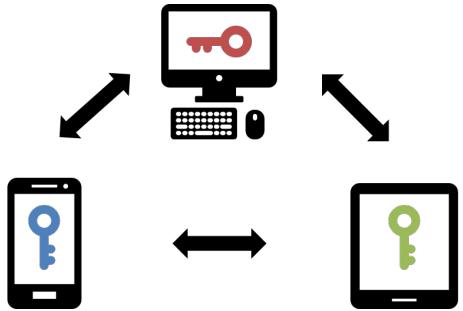


Threshold computation

Practical problems of threshold protocols

- Protocol selection
  - Several protocols exist for each signature signature type
  - Computation (operations, complexity, …)
  - Communication (rounds, bandwidth, reliable broadcast, …)
  - Security assumptions (standard vs less conventional)
- Lack of (open-source) implementations
  - Limited testing and security evaluations
  - Restrictive licenses

Single party computation

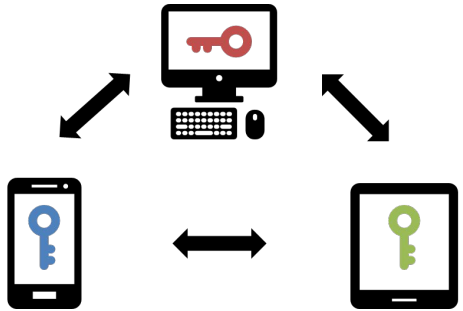


Threshold computation

Practical problems of threshold protocols

- Protocol selection
  - Several protocols exist for each signature signature type
  - Computation (operations, complexity, …)
  - Communication (rounds, bandwidth, reliable broadcast, …)
  - Security assumptions (standard vs less conventional)
- Lack of (open-source) implementations
  - Limited testing and security evaluations
  - Restrictive licenses
- Lack of standardization
  - NIST initiated efforts to standardize threshold schemes [NIST23]
  - IETF is currently standardizing FROST scheme [IETF23]

Single party computation

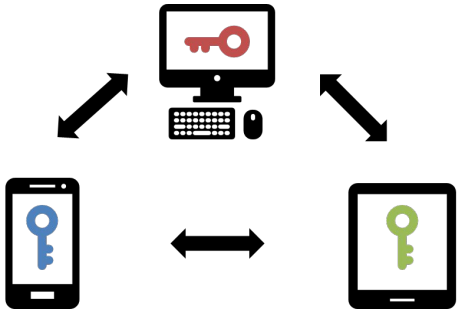


Threshold computation

Practical problems of threshold protocols

- Protocol selection
  - Several protocols exist for each signature signature type
  - Computation (operations, complexity, …)
  - Communication (rounds, bandwidth, reliable broadcast, …)
  - Security assumptions (standard vs less conventional)
- Lack of (open-source) implementations
  - Limited testing and security evaluations
  - Restrictive licenses
- Lack of standardization
  - NIST initiated efforts to standardize threshold schemes [NIST23]
  - IETF is currently standardizing FROST scheme [IETF23]
- Secure handling with secret shares

Single party computation

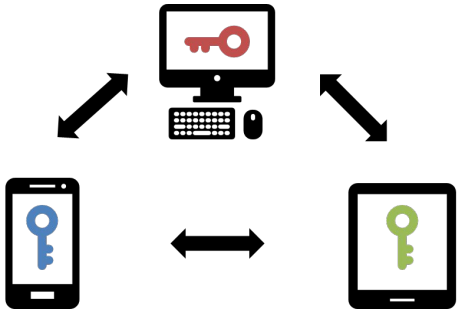


Threshold computation

Secure handling with secret shares

- It is difficult to store and use secrets securely

Single party computation

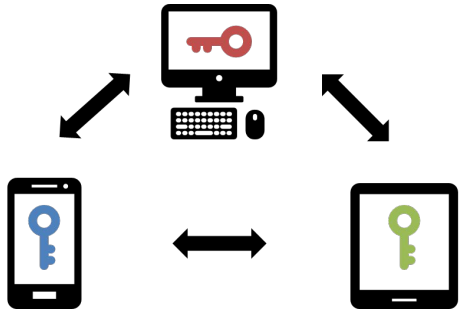


Threshold computation

Secure handling with secret shares

- It is difficult to store and use secrets securely
- Consumer devices contain secure hardware modules
  - Android Strongbox, iOS Enclave, TPM, …

Single party computation

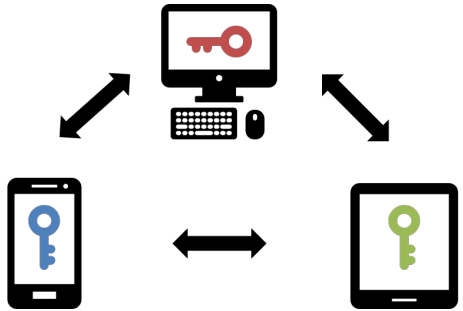


Threshold computation

Secure handling with secret shares

- It is difficult to store and use secrets securely
- Consumer devices contain secure hardware modules
  - Android Strongbox, iOS Enclave, TPM, …
  - But they do not support threshold protocols

Single party computation



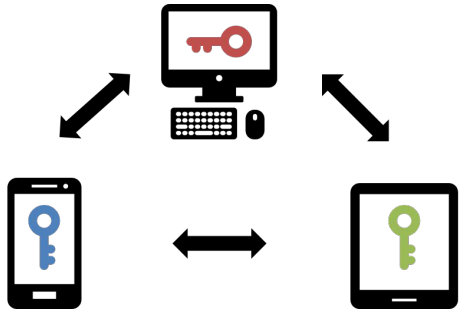
Threshold computation

Secure handling with secret shares

- It is difficult to store and use secrets securely
- Consumer devices contain secure hardware modules
  - Android Strongbox, iOS Enclave, TPM, …
  - But they do not support threshold protocols

- Threshold protocols are not standardized

Single party computation



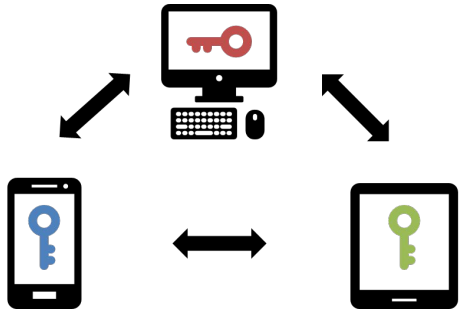
Threshold computation

Secure handling with secret shares

- It is difficult to store and use secrets securely
- Consumer devices contain secure hardware modules
  - Android Strongbox, iOS Enclave, TPM, …
  - But they do not support threshold protocols

Single party computation

- Threshold protocols are not standardized
- Secure hardware may not be able to run the protocols
  - Hardware constraints
  - Limited interface and programmability



Threshold computation

Research question

Can state-of-the-art threshold protocols be computed efficiently on current secure hardware?

Research question

Can state-of-the-art threshold protocols be computed efficiently on current **JavaCards**?

Research question

Can state-of-the-art threshold protocol **FROST** be computed efficiently on current **JavaCards**?

FROST

- State-of-the-art threshold Schnorr signature scheme [KG21]

FROST

- State-of-the-art threshold Schnorr signature scheme [KG21]
- The standardization draft defines `FROST(E, H)` [IETF23]
  - Parameterizable by an elliptic curve and a hash function
  - 5 ciphersuites specified (secp256k1, P-256, Ed25519, ristretto255, Ed448)

FROST

- State-of-the-art threshold Schnorr signature scheme [KG21]
- The standardization draft defines `FROST(E, H)` [IETF23]
  - Parameterizable by an elliptic curve and a hash function
  - 5 ciphersuites specified (secp256k1, P-256, Ed25519, ristretto255, Ed448)
- Two communication rounds
  - The first is offline (precomputable)
  - The second depends on the message

FROST

- State-of-the-art threshold Schnorr signature scheme [KG21]
- The standardization draft defines `FROST(E, H)` [IETF23]
  - Parameterizable by an elliptic curve and a hash function
  - 5 ciphersuites specified (secp256k1, P-256, Ed25519, ristretto255, Ed448)
- Two communication rounds
  - The first is offline (precomputable)
  - The second depends on the message
- The computation requires
  - Modular arithmetic (addition, subtraction, multiplication, inversion)
  - Elliptic curves (point addition, scalar multiplication)
  - Hash function

JavaCard smartcards

- The most widely used secure hardware (20B+ devices sold)

JavaCard smartcards

- The most widely used secure hardware (20B+ devices sold)
- Versatile (usable with other devices – NFC or USB reader)

JavaCard smartcards

- The most widely used secure hardware (20B+ devices sold)
- Versatile (usable with other devices – NFC or USB reader)

- Generic CPU
  - Programmable with Java-like code that gets executed in JCVM
  - Insufficient performance for current public-key cryptography

JavaCard smartcards

- The most widely used secure hardware (20B+ devices sold)
- Versatile (usable with other devices – NFC or USB reader)

- Generic CPU
  - Programmable with Java-like code that gets executed in JCVM
  - Insufficient performance for current public-key cryptography
- Hardware accelerators
  - Accessible only via high-level JC API algorithms (RSA, DH, ECDH, …)
  - Lack of low-level operation access (addition, multiplication, EC addition)
  - Repurposing algorithms to perform low-level operations

JCMathLib

- A JavaCard library providing modular arithmetic and EC operations [MS20]

JCMathLib

- A JavaCard library providing modular arithmetic and EC operations [MS20]
- Implemented in 2017 for JC API 3.0.1
  - Lacking improvements from newer API versions
  - Missing support for latest smartcards

JCMathLib

- A JavaCard library providing modular arithmetic and EC operations [MS20]
- Implemented in 2017 for JC API 3.0.1
  - Lacking improvements from newer API versions
  - Missing support for latest smartcards
- We extended the library
  - Support for new cards
  - Faster operations on new cards
    - Modular multiplication (4.5x speedup)
    - Accelerated point addition (7x speedup)
    - Fully accelerated scalar multiplication (4x speedup)
  - Added new operations
    - Combined point addition and multiplication (useful for FROST)
    - Point encoding and decoding

FROST on JavaCards

JCFROST

- Open-source FROST protocol implementation for JavaCards
  - Implements `FROST(secp256k1, SHA-256)`
  - Compliant with the current version of the FROST standardization draft [IETF23]
  - Relies only on public JavaCard API! (thanks to JCMathLib)

JCFROST

- Open-source FROST protocol implementation for JavaCards
  - Implements `FROST(secp256k1, SHA-256)`
  - Compliant with the current version of the FROST standardization draft [IETF23]
  - Relies only on public JavaCard API! (thanks to JCMathLib)

- Optional optimizations (keep compatibility)

JCFROST

- Open-source FROST protocol implementation for JavaCards
  - Implements `FROST(secp256k1, SHA-256)`
  - Compliant with the current version of the FROST standardization draft [IETF23]
  - Relies only on public JavaCard API! (thanks to JCMathLib)

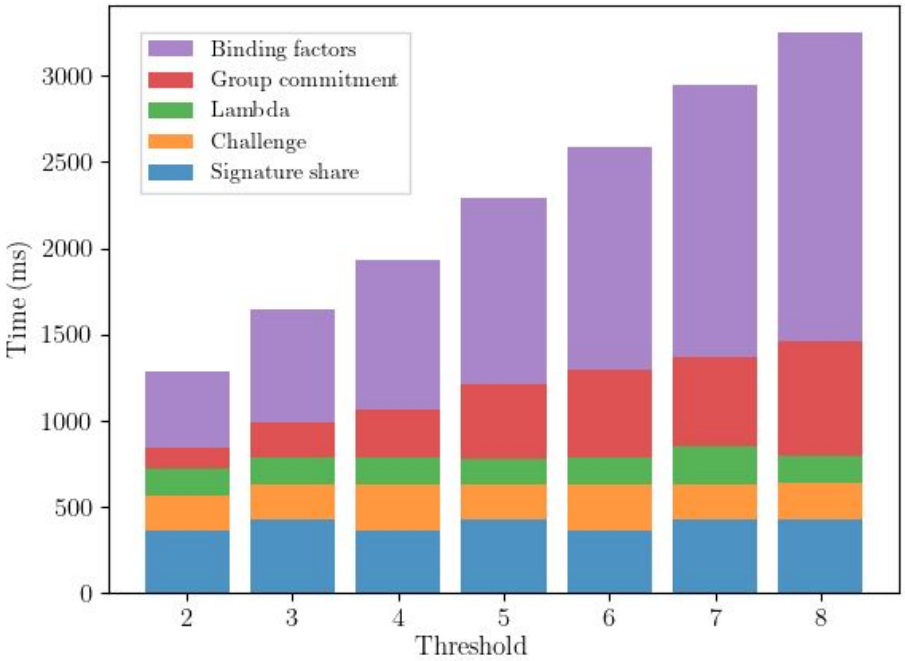
- Optional optimizations (keep compatibility)
  - Point decompression by host
    - No additional trust put on the host as ECPoint checks that points lie on the curve
    - Suitable for point transformations of non-Weierstrass curves (e.g., Ed25519, …)

JCFROST

- Open-source FROST protocol implementation for JavaCards
  - Implements `FROST(secp256k1, SHA-256)`
  - Compliant with the current version of the FROST standardization draft [IETF23]
  - Relies only on public JavaCard API! (thanks to JCMathLib)

- Optional optimizations (keep compatibility)
  - Point decompression by host
    - No additional trust put on the host as ECPoint checks that points lie on the curve
    - Suitable for point transformations of non-Weierstrass curves (e.g., Ed25519, …)
  - Constrained lambda computation ($n \leq 12$ or $n \leq 7$)
    - Intermediate computations can fit in int or short (depending on the constraint)
    - Allows for significant performance improvement over BigInteger-based version

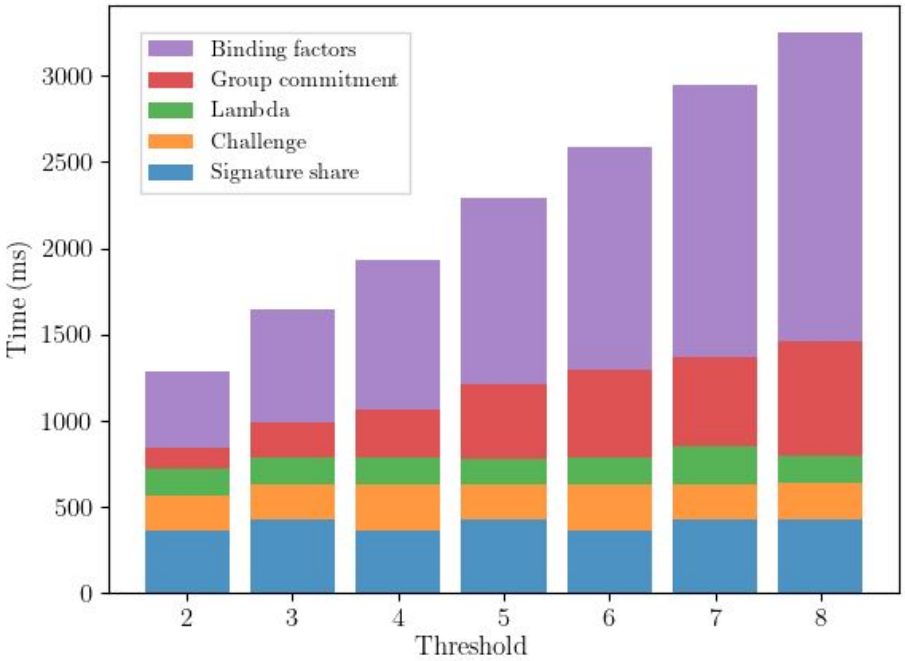
Performance results



NXP J3R200 (2020)

- t < 5 can be computed under 2s

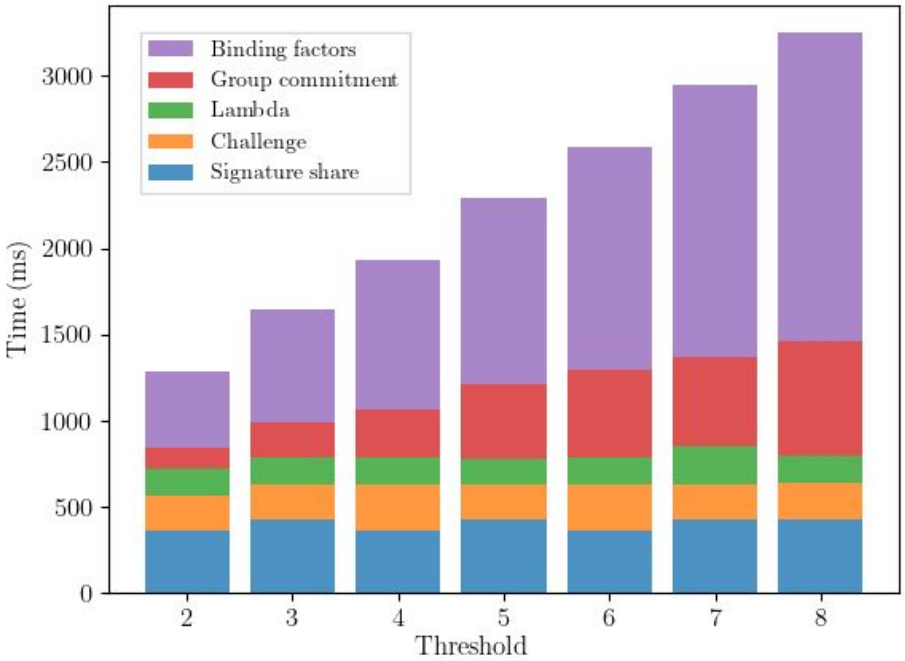
Performance results



NXP J3R200 (2020)

- t < 5 can be computed under 2s
- Linear growth with threshold

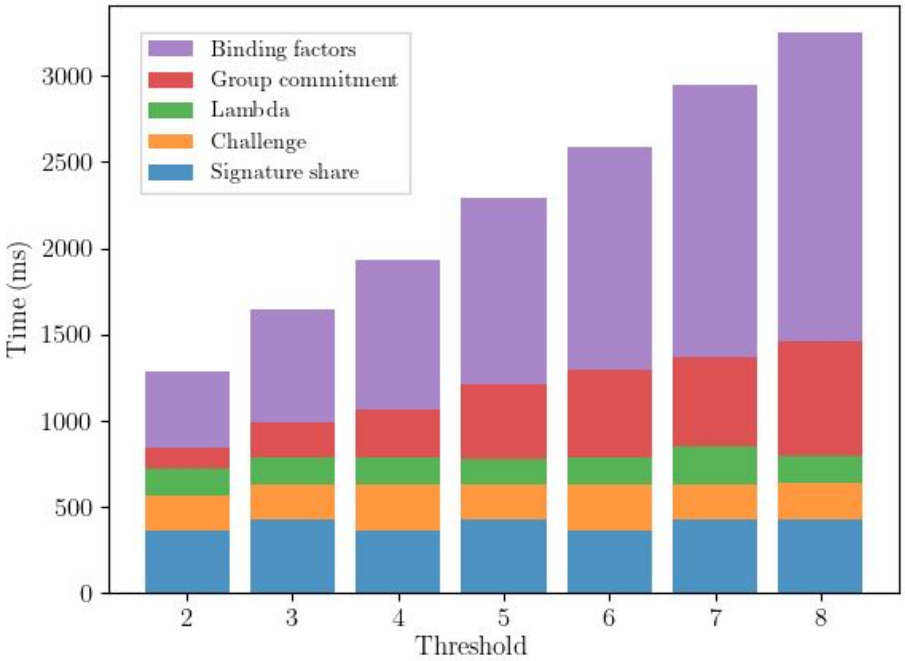
Performance results



NXP J3R200 (2020)

- t < 5 can be computed under 2s
- Linear growth with threshold
- Group commitment performs on the platform's limits

Performance results



NXP J3R200 (2020)

- t < 5 can be computed under 2s
- Linear growth with threshold
- Group commitment performs on the platform's limits
- Other operations could be notably faster with low-level access to the platform

Demonstration

Conclusion

- Can threshold protocol FROST be computed efficiently on current JavaCards?
  - Yes, even when relying only public JavaCard API
  - Performance improvements possible using proprietary low-level interface

Conclusion

- Can threshold protocol FROST be computed efficiently on current JavaCards?
  - Yes, even when relying only public JavaCard API
  - Performance improvements possible using proprietary low-level interface
- The positive result can generalize to similar EC-based schemes
  - CoSi [Syt+17], MuSig2 [JRS21], SimpleTSig [CKM22], Lindell22 [Lin22]

Conclusion

- Can threshold protocol FROST be computed efficiently on current JavaCards?
  - Yes, even when relying only public JavaCard API
  - Performance improvements possible using proprietary low-level interface
- The positive result can generalize to similar EC-based schemes
  - CoSi [Syt+17], MuSig2 [JRS21], SimpleTSig [CKM22], Lindell22 [Lin22]
- On the contrary
  - ECDSA-based protocols currently seem too complex for this approach
  - Post-quantum signatures are mostly too demanding even in the single-party case

Thank you for your attention

JCFROST implementation

https://github.com/crocs-muni/JCFROST

JCMathLib library

https://github.com/OpenCryptoProject/JCMathLib

References

[KG21] Komlo, Chelsea, and Ian Goldberg. "FROST: flexible round-optimized Schnorr threshold signatures." Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers 27. Springer International Publishing, 2021.

[NIST23] NIST. NIST First Call for Multi-Party Threshold Schemes." https://csrc.nist.gov/publications/detail/nistir/8214c/draft. 2023.

[IETF23] Connolly, Deidre, Chelsea Komlo, Ian Goldberg, Christopher A. Wood. "Two-Round Threshold Schnorr Signatures with FROST." IETF standardization draft. Draft version 14. 2023. https://datatracker.ietf.org/doc/draft-irtf-cfrg-frost/.

[MS20] Mavroudis, Vasilios, and Petr Svenda. "JCMathLib: wrapper cryptographic library for transparent and certifiable JavaCard applets." 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). IEEE, 2020.

References

[Syt+16] Syta, Ewa, et al. "Keeping authorities" honest or bust" with decentralized witness cosigning." 2016 IEEE Symposium on Security and Privacy (SP). Ieee, 2016.

[JRS21] Nick, Jonas, Tim Ruffing, and Yannick Seurin. "MuSig2: simple two-round Schnorr multi-signatures." Annual International Cryptology Conference. Cham: Springer International Publishing, 2021.

[CKM22] Crites, Elizabeth, Chelsea Komlo, and Mary Maller. "How to prove Schnorr assuming Schnorr: security of multi-and threshold signatures." Cryptology ePrint Archive (2021).

[Lin22] Lindell, Yehuda. "Simple three-round multiparty schnorr signing with full simulatability." Cryptology ePrint Archive (2022).