24.07.23 NIST Signature and Halo2 Benchmark

Xun Zhang Bingsheng Zhang
Zhejiang University, CHN
22221024@zju.edu.cn bingsheng@zju.edu.cn

July 23 2024

1 NIST Multi-party Threshold Cryptography

NIST is calling for Multi-Party Threshold Schemes, see **NIST IR 8214C (Initial Public Draft), NIST First Call for Multi-Party Threshold Schemes.** https://csrc.nist.gov/pubs/ir/8214/c/ipd. NIST

Below is the overview of Multi-Party Threshold Cryptography MPTC, see https://csrc.nist.rip/Projects/threshold-cryptography: MPTC

The multiparty paradigm of threshold cryptography enables a secure distribution of trust in the operation of cryptographic primitives. This can apply, for example, to the operations of key generation, signing, encryption and decryption.

This project focuses on threshold schemes for cryptographic primitives: using a "secret sharing" mechanism, the secret key is split across multiple "parties", such that, even if some (up to a threshold f out of n) of these parties are corrupted, the key secrecy remains uncompromised, even during the cryptographic operation that depends on the key. This approach can be used to distribute trust across various operators, and is also useful to avoid various single-points of failure in the implementation.

The multi-party threshold cryptography project will consider devising guidelines and recommendations pertinent to threshold schemes that are interchangeable (in the sense of NISTIR 8214A, Section 2.4) with ECDSA signing, EdDSA signing, RSA signing and decryption, and AES encryption/decryption, and their respective distributed key-generation. For example, a signature produced by a threshold scheme should be verifiable by the same algorithm as used for conventional signatures.

2 Discussion about Schnorr

Most contents is from the slides *'Recent Developments on Multi-Party Schnorr Signatures'* of Elizabeth Crites, University of Edinburgh.(Nov. 24, 2022). see https://elizabeth-crites.github.io/london_crypto_day.pdf Schnorr

2.1 Why Schnorr?

- RSA, ECDSA, EdDSA (Aug.'22) standardized through NIST.
- EdDSA is deterministic version of Schnorr.
- RSA signatures are large (6x ECDSA/EdDSA).
- ECDSA requires nonce inversion and other complexities. And no security reduction like Schnorr -¿ DL + ROM.
- BLS requires bilinear pairings (no NIST standardization).

2.2 Why Now?

- Bitcoin moved from ECDSA to Schnorr (Nov.'21).
- MuSig2 [NRS21] / FROST [KG20] proposed to secure cryptocurrency wallets.
- FROST IETF draft [CKGW22], 9+ implementations.
- NIST call for threshold EdDSA/Schnorr threshold signatures [BD22]. EdDSA not verifiably deterministic so Schnorr can be used instead.

2.3 Threshold Schnorr

The terminology 'Threshold' here is for such scenario: there is only **ONE** public key, and the holder will use Shamir method to secret-share the private key. And also the nonce **r**. And finally it will collect all the signatures and verify the threshold schnorr signature.

So this setting is different from what we want(many signers holds their own pk/sk).

3 Schnorr Multi-signature with Merkle Tree

We implemented the Schnorr multi-signature scheme with a Merkle Tree, which is used for membership-proof. The workflow is same as we describe in the last week slides, we replace the encode method with a merkle tree root:

$a_i = H(L||X_i)$ for all X_i, where $L = ROOT(X_1, X_2, ..., X_n, NX_1, NX_2, ..., NX_m)$

Then we prove that all the public keys we used to produce aggregated public key is a member of this Merkle Tree(corresponding to this merkle root).

Follwing is the results of proving time:

Setting	Proving Time
k = 14, 8_of_16	1.4335s
k = 15, 16_of_32	2.4763s
k = 16, 32_of_64	4.5464s
k = 18, 64_of_128	16.384s
k = 19, 128_of_256	31.089s

Table 1: Proving time of Schnorr Multi-signature with Merkle Tree

For example, **8** is the number of public keys participate in the signing phase, and **16** is the number pf whole public key set, which is commited as a merkle tree root.

4 Signature Scheme behind the top cryptocurrencies

Signature Scheme	Crypto
Schnorr	Bitcoin
EdDSA	XRP, Cardano, Stellar, Monero, Cosmos
ECDSA	Ethereum, XRP, Litecoin, Binance Coin, Tron
ERC20	USDT, Chainlink, USD Coin, OKB
BLS	Ethereum 2.0, Cardano

Table 2: Siganture Scheme behind the top cryptocurrencies

5 BLS Multi-signature

We benchmarked the BLS Multi-signature implementation in the halo2-lib, with public key aggregation, but not hash function(which means we use a random input as msghash value).

And the halo2-lib crate use two-time pairing(one is inside the halo2-lib, the other is a outer implementation) to illustrate the correctness of (their)pairing. So it can be optimized in further.

Here are the results:

degree	advice	lookup	num_aggregation	proof_time	proof_size	verify_time
17	25	3	2	17.4515s	11520	72.7712ms
17	25	3	20	18.5408s	12224	73.2885ms
17	25	3	200	22.3638s	15008	89.0890ms
17	25	3	2000	67.0237s	45952	286.8686ms
17	25	3	4000	116.4392s	79680	327.1002ms
17	25	3	6000	165.0348s	113760	551.4775ms
17	25	3	8000	210.1191s	147840	664.9270ms
17	25	3	10000	256.9934s	181920	841.5073ms

Table 3: Bn254 BLS Multi-Signature Verification Cost

This result is running on a linux server. With a Intel Xeon Silver 4241(48cores) @3.200GHz, and 128G RAM, Ubuntu 20.04.6 LTS.

6 A Brief Conclusion about Signatures in Halo2

Here we list some benchmark results of various signatures. Note that the performance are under different settings(even code).

Signature	curve	Proving Time
EDDSA	ed25519	10.1348s
ECDSA	secp256k1	3.5269s
Schnorr	BLS/Jubjub	0.4s
BLS	Bn254	20.4768s

Table 4: Caption

By the way, if we implement(have implemented actually) the **EdDSA** signature in Inigo's code('sidechains-zk'), it is also very fast. Because the verification process is totally same as Schnorr signature.