Technical whitepaper

Shard X - blockchain API with MPC signing

Nikita Lesnikov (nikita@shardx.io)

Abstract. Peer-to-peer network interaction and private key management are two common yet hard problems that most blockchain projects need to tackle. Shard X is a uniform API striving to solve both problems across many blockchains. This document outlines the high level design of this API.

KEY WORDS

1. Crypto asset 2. Blockchain 3. Multiparty computation

1. Established practices

Crypto assets are trustless by design, which means they can't use the established authentication mechanisms (for instance, email & password) and fallbacks (password reset via second factor), as there is no central trusted party for that. They rely on public key cryptography instead. Specifically, a secret known as a *private key* corresponds to an *address* in a deterministic way; knowledge of it equates to having the ability to transact without the consent of the true owner; if the private key is completely lost, funds cannot be recovered. These are two very undesirable traits that make key management for crypto assets a difficult problem. The more copies of the private key are stored, the greater the potential attack surface, yet backups are an absolute must.

There are many established practices for dealing with this trade-off, each with unique advantages and disadvantages designed for varying use cases. For instance, if funds are accessed infrequently and very high degree of security is sought for, *cold storage* is the traditional answer - private keys are stored offline (sometimes custodians go as far as physically keeping them in bank vaults), and all the signing happens on airgapped[1] devices. But obviously this scheme is inadequate in case significant throughput or automation is required - cold storage usually takes hours to days to access the funds and often involves manual processing.

For crypto asset applications requiring any kind of automatic processing, private keys must reside on a networked device in order to sign and broadcast transactions. Thus a hacker has a very serious incentive to break into the system and steal the funds. Crypto asset exchanges are a prime example of this case, as they need a significant balance on a hot wallet (in order to seamlessly process withdrawals) and at the same time should be completely automated, which prevents using cold storage. In fact, theft of hot wallet private keys is the number one root cause of exchange hacks (Fig. 1). Another valid example of this use case is delegated staking or baking of certain crypto assets.

Some blockchains implement *multisig* functionality, where signatures from multiple private

[1] not connected to any computer network

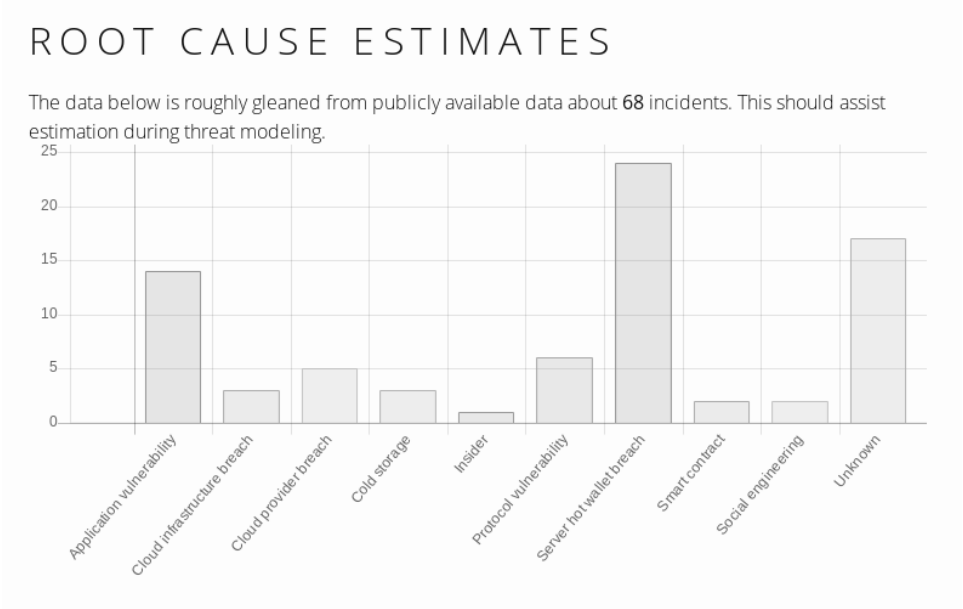

Fig. 1. Taken from the Blockchain Graveyard

keys are required in order to move funds from the account. This improves security by adding redundancy (funds are safe in case at least some of the private keys are *not* compromised), but carries the cost of higher transaction fees (extra space is required to store the additional signatures on the blockchain), interoperability issues (for instance, Bitcoin generally requires P2SH/P2WSH addresses in order to implement multisig, Ethereum requires the use of smart contracts), and some loss of privacy (multisig transactions look very different compared to "regular" ones on the blockchain).

Orthogonally to the above, it is generally believed that HSMs[2] are preferable to software only solutions. Typical HSM would store private key internally and only expose a very limited signing interface. Yet they are by no means a silver bullet, for several reasons:

- HSM can malfunction (reports of "bricked" devices are not uncommon)
- there are very feasible supply chain attacks which involve flashing malicious firmware to HSMs at the factory, which either allows an attacker with physical access to the device to extract the private key directly, or leak it via generated signatures even when used in airgapped setting [6]

Blockchain projects which involve dealing with significant amounts of crypto assets employ some combination of the above to manage their assets, making many compromises on their way. Shard X strives to address the common pain points by offering an all-in-one API service combining multi-party computation singing and node hosting. Both are elaborated below.

2. Multi-party computation

Multi-party computation (MPC) is a branch of cryptography which deals with scenarios of multiple distrustful parties performing a single computation. There is a vast corpus of recent research into applying MPC techniques to digital signing, with immediate applications in crypto

[2] Hardware Security Modules

assets.

More specifically, MPC can be used to provide a *threshold signature* functionality in the following way:

(1) Several parties follow a specific protocol to generate multiple independent *secrets*, which are never shared (and should not be).

(2) These secrets are used in another protocol to produce a single digital signature.

While the value offering of MPC signing sounds very similar to that of a multisig address, the key distinction is that only a single digital signature is produced as a result, without requiring special support from the underlying blockchain. Given that most current blockchains use a small common set of cryptographic primitives (Table 1), the technology is broadly applicable.

Table 1. Elliptic curves of different blockchains

Blockchain	Elliptic curve
Bitcoin	*secp256k1*
Ethereum	*secp256k1*
Tron	*secp256k1*
EOS	*secp256k1*
Ripple	*ed25519*
Tezos	*secp256k1*[a]
ZCash	*JubJub*

[a] can use *ed25519* & *NIST P256* as well

An MPC signing implementation over a few common elliptic curves and signing algorithms may cover multiple blockchains simultaneously, providing the same level of security and interoperability. It is important to note that such uniformity is a novel thing. For instance, native token transfers on Ethereum do not support multisig, hence smart contracts are required to provide similar functionality, which is a solution with its own set of compromises. Bitcoin does support multisig natively, but the implementation suffers from high fees and is restricted to a subset of address types. MPC signing compares favorably to both approaches, being cheaper, less restricted in application, and working in essentially the same way over this two very different blockchains. What is more important, however, is that MPC signing may be used well beyond Bitcoin & Ethereum, in essentially unchanged way.

On a side note, it was shown that MPC signing may be used as a building block to provide "smart contract" like functionality over non-Turing-complete blockchains like Bitcoin [4].

This concludes the explanation of the alleged benefits of this approach. Below we explain how MPC signing helps Shard X achieve improved security & uniformity of private key management.

3. 2-of-2 MPC signing

The simplest, yet arguably most useful application of MPC signing is 2-of-2 threshold scheme [3], where a single address is controlled by two secrets, both of which are required to produce a signature. On a first attempt to instantiate such a scheme one might envision splitting a private key into parts (using Shamir Secret Sharing Scheme [5], for instance) and recombining them on each signing attempt. An important distinction of MPC signing, however, is that private key is

never instantiated explicitly.

Key generation happens according to Fig. 2 and is sequenced as follows:

(1) Both parties come up with random secrets on their own.
(2) One of the parties (which we designate *Client* for the reasons apparent below) sets up a homomorphic encryption channel.
(3) Both parties agree on a common public key.
(4) At this point, an MPC signing *address* is effectively created - there are two parties holding pieces of secret information that can be used to reconstruct the private key.

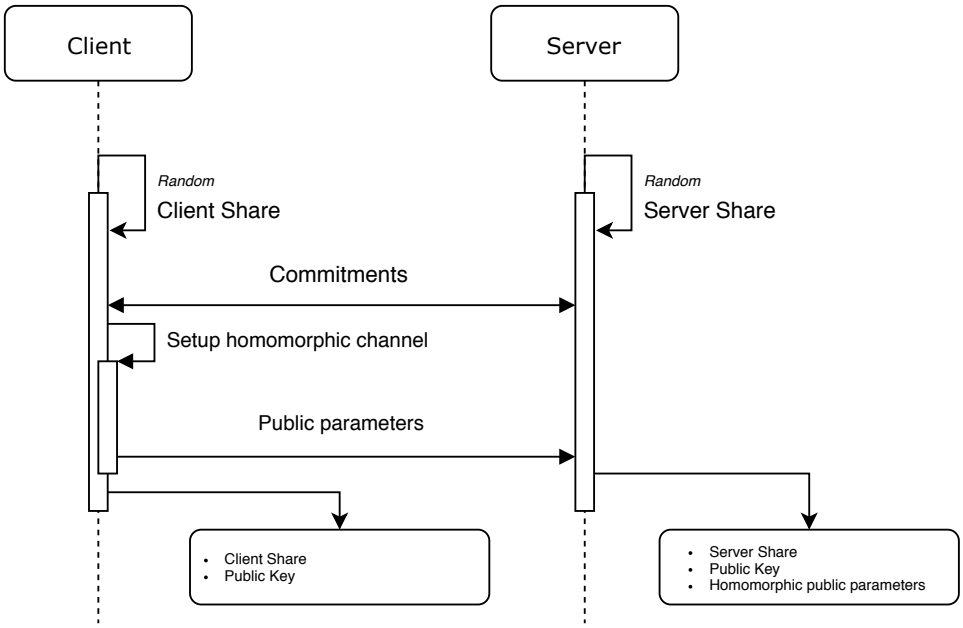

Fig. 2. 2-of-2 MPC ECDSA key generation

The defining feature of MPC signing, however, is that the private key never has to be reconstructed - respective secrets never leave the hosts they were generated on. So in case a signature needs to be produced with a generated address, the protocol from Fig. 3 has to be used. First, the Client and Server share the message to be signed, as well as agree on a common random point (in zero-knowledge, using a protocol that is very similar to Diffie-Hellman [1]). Then the Server uses its secret to create a partial signature that is encrypted using a homomorphically additive[3] scheme (parameters of which were determined in the key generation phase). Next, the *Client* finalizes the computation by decrypting the partial signature and finalizing it using its secret. Please note that in this protocol only the *Client* has the ability to produce a final signature.

This architecture has a lot of convenient applications in practice. For instance, imagine that some service needs to manage a variety of crypto asset wallets. Whenever it needs a fresh address, it invokes the key generation protocol, creating a Client secret locally and storing it into the local database (the corresponding server secret is created at the same time by MPC signing API according to Fig. 2). The client secret is less sensitive than a raw private key in a sense that it's

[3] *Homomorphically additive* means that the encryption scheme can be used to add encrypted values together or multiply them by a scalar, all without decrypting them first

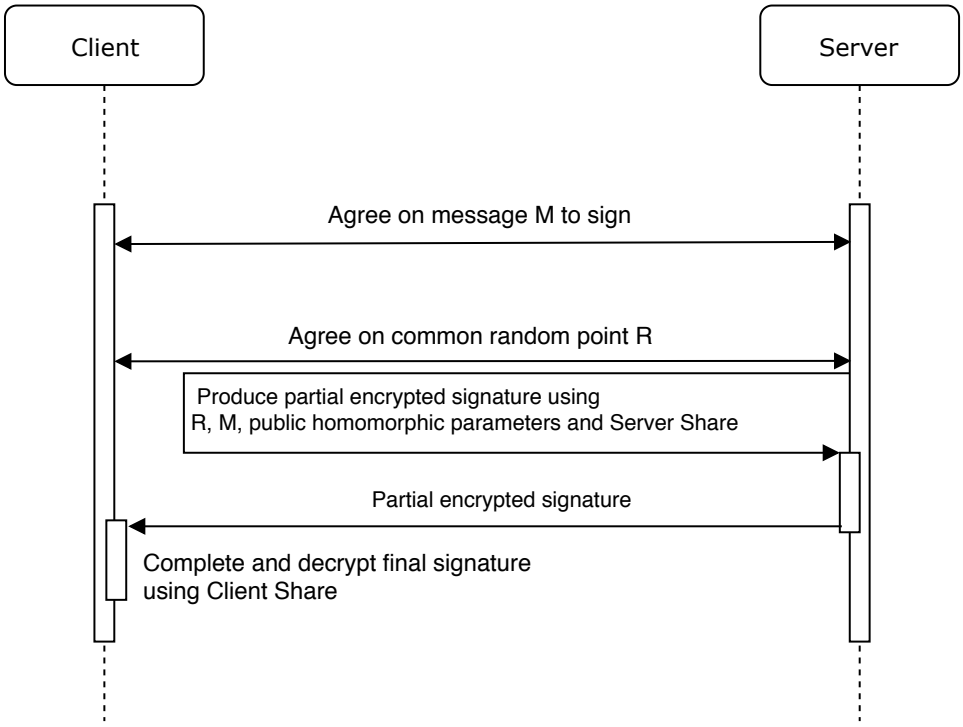

Fig. 3. 2-of-2 MPC ECDSA signing

not self-sufficient - in case its compromised, it is of little use as an attacker is unable to produce a valid signature with it. In order to perform any signing operation, an MPC signing API has to be used, subject to IP whitelisting, rate & value limiting and other fraud detection mechanisms. This effectively places a large restraint of the hackers ability to succeed. At the same time, an MPC signing API doesn't hold any custody - it's only responsibility is to finalize the *partial encrypted* signatures, so in case the service provider is compromised, funds in MPC wallets are unaffected (provided the Client that is able to finalize the signatures was not compromised as well).

It's worth reiterating that the result of MPC signing doesn't differ from any regular signature, and that makes such an API a drop-in replacement for many applications. Given the significance of the benefits, 2-of-2 MPC signing is the approach that Shard X uses *by default*.

What are the drawbacks? First and foremost, the scheme can't be deployed offline - any signature has to be constructed by both parties in several rounds, and that communication can't be eliminated. But a fully offline (or unidirectional) application is already covered by cold storage solutions. Thus a more significant drawback is not the online requirement per se, but the reliance on the other party availability - it is obvious from the scheme description that no signature can be produced if the Server goes offline, disappears entirely or becomes malevolent. To protect from such an occasion, Shard X incorporates the *rescue bundle* functionality - an ability to create a set of pre-signed transactions moving funds to the designated address, which can be broadcasted by Client alone. A limited set of such transactions can enable the Client to withdraw an arbitrary amount to the predetermined address, providing unilateral access to funds. Since that address is selected and nominated by the Client and cannot be changed without the service provider, it negates the risk of the hackers ability to use the *rescue bundle* as an attack surface. For instance,

in Ethereum such a bundle may be constructed by pre-signing $\log(n)$ transactions per nonce with power-of-two amounts for a consecutive span of nonces (see Fig. 4). Please note that the very ability to create such a bundle unilaterally stems from the fact that only Client has access to the decrypted final signature.

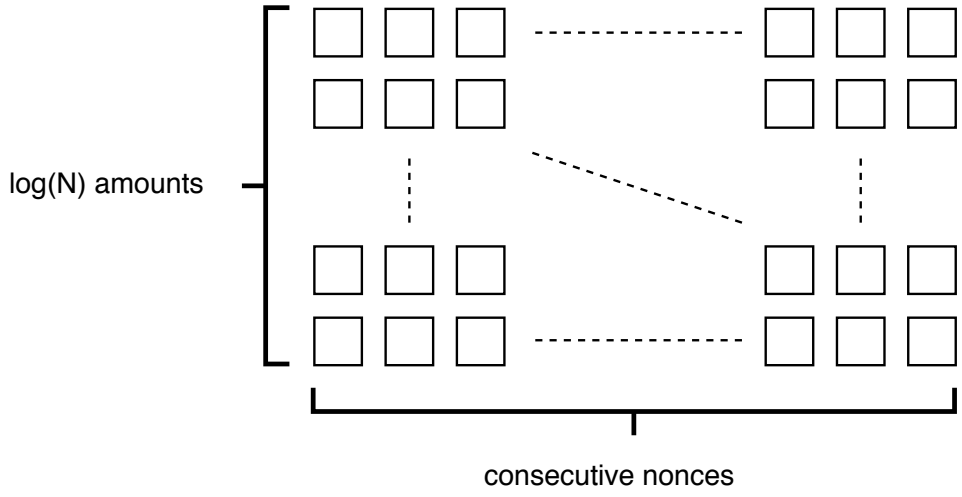

Fig. 4. Shard X rescue bundle (for Ethereum)

4. Shard X signing architecture

At a high level, MPC signing is a relatively simple technology, yet its implementation requires great attention to detail, especially regarding the design of communication between Client and Server. Care must be taken to perform all the required cryptographic commitments and validate the *encrypted* data. For maximum security, all communication has to be performed in zero knowledge, which means that an eavesdropper cannot gain *any* insight from the full transcript. Additionally, the Client and Server secrets should never be transmitted.

To satisfy these requirements, Shard X introduces the concept of an API client (see Fig. 5) for dealing with sensitive data. API client can be used either as a WebAssembly library or a standalone gateway microservice. The first scenario is tailored towards Node.js deployments, web applications and browser extensions, while the second one aims to be deployed as a part of some backend infrastructure. Gateway microservice communicates with Shard X servers on its own and is accessed externally via RESTful HTTP API, which makes it easy to integrate into almost any environment. WebAssembly library has to perform the communication via the callbacks provided by the developer, which are however rather generic and abstract away the protocol details.

In both cases the general idea stays the same - there is a Shard X component on Client side which encapsulates away all the MPC know-how. Both WebAssembly library and the standalone microservice are built from the same Rust codebase and are open source.

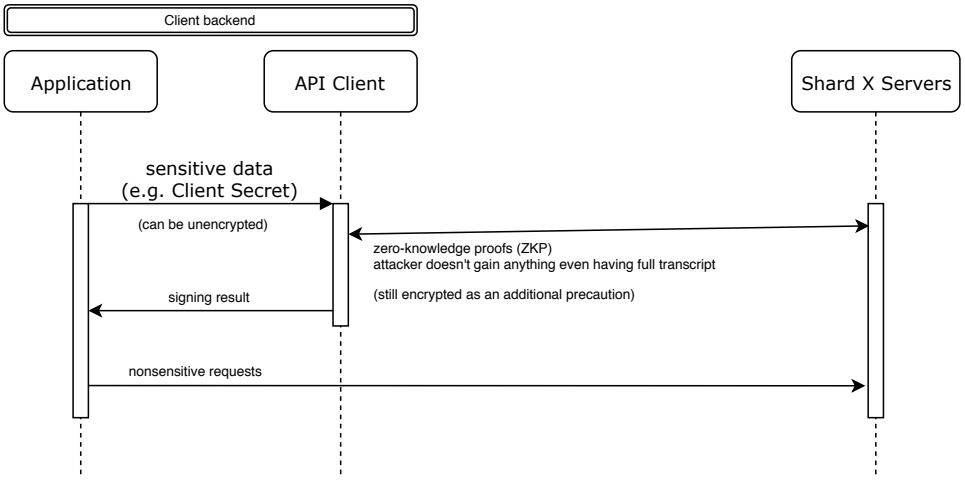

Fig. 5. Shard X API Client concept

5. Node hosting service

While MPC key management is a useful primitive on its own, one of its biggest applications is dealing with crypto assets. For that a client application needs to query the state of the respective blockchain, assemble, sign and broadcast transactions. Most of this functionality is not rocket science, yet have proven quite challenging for many developers - blockchain nodes have to be deployed redundantly in several locations for adequate availability, take time to sync from scratch, have to be updated when forks happen.

To provide a one-stop-solution for programmatically accessing crypto assets, Shard X hosts its own nodes. A developer is provided a RESTful API facade, which transparently caches and load balances client requests over available nodes. Shard X node hosting services is different from the competitors in the following ways:

- Uniform least-common-denominator functionality - basic routines like querying balances, signing and broadcasting asset transfers are abstracted away from an underlying blockchain. Thus if an application doesn't require any specific functionality, covering new currencies becomes trivial.
- Full functionality is exposed in case the developer needs it: vendor-specific requests, archive node functionality, debug tracing, etc.
- Shard X aims to fully support Bitcoin and Ethereum at launch, but it doesn't stop there - the goal is to support all significant blockchains.

MPC is a key ingredient here, because it allows to manage keys securely with threshold schemes without resorting to protocol-specific approaches. This should eventually provide unprecedented breadth of supported blockchains simultaneously with significantly elevated security.

6. Directions of future improvement

At launch Shard X is delivering a minimum viable product of 2-of-2 MPC signing for Bitcoin and Ethereum, with API client as two deliveries: a WASM library and a standalone RESTful

microservice. Following from here, major directions for improvement are:

- More generic *k*-of-*n* threshold schemes using more computationally heavy protocols [2]
- Wider blockchain support (Bitcoin & Ethereum forks, EOS, Tezos)
- More robust unilateral access alternatives to *rescue bundle*, for instance an ability for vast majority of Shard X users to collude and recover their private keys in case the service is gone for whatever reason

7. References

[1] W. Diffie and M. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (Nov. 1976), pp. 644–654. ISSN: 0018-9448. DOI: 10.1109/TIT.1976.1055638.

[2] Rosario Gennaro and Steven Goldfeder. "Fast Multiparty Threshold ECDSA with Fast Trustless Setup". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada: ACM, 2018, pp. 1179–1194. ISBN: 978-1-4503-5693-0. DOI: 10.1145/3243734.3243859. URL: http://doi.acm.org/10.1145/3243734.3243859.

[3] Yehuda Lindell. *Fast Secure Two-Party ECDSA Signing*. Cryptology ePrint Archive, Report 2017/552. https://eprint.iacr.org/2017/552. 2017.

[4] Aniket Kate Pedro Moreno-Sanchez. *Scriptless Scripts with ECDSA*. 2018.

[5] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: http://doi.acm.org/10.1145/359168.359176.

[6] Stepan Snigirev. *Hardware wallets can be hacked, but this is fine*. https://medium.com/cryptoadvance/hardware-wallets-can-be-hacked-but-this-is-fine-a6156bbd199.