TLDR

: We propose a generic approach to safely deploy cryptographic primitives at the protocol-level. We use canary contracts to detect unsafe primitives, and to automatically deploy safer cryptographic backups.

Background

We want the Ethereum protocol to endure on the order of centuries yet cryptographic primitives break down on the order of decades. One generic approach to address this is “abstraction” where cryptographic choices are pushed away from the protocol layer towards users and the application layer.

Unfortunately cryptographic abstraction has its limits. In some cases we may want the protocol to enforce homogeneity across all participants (e.g. it’s useful for all Merklelised data structures to use the same hash function), and in other cases one particular cryptographic construct has a set of features that makes it uniquely appropriate for a given task, and not making use of that construct could be a wasted opportunity.

The “wasted opportunity” aspect resonates with considerations around quantum security. At this point the quantum era does seem inevitable and many primitives (e.g. ECDSA, RSA, BLS signatures, SNARKs) are known to be “pre-quantum”, i.e. not be post-quantum secure. However, the dawn of the post-quantum era may not happen for another decade, and the next decade in cryptoland is particularly critical. This makes totally avoiding pre-quantum crypto at the protocol-level a potential strategic mistake.

Below are four examples of fancy/safe pairs of primitives where the fancy (pre-quantum) version is genuinely more powerful than the safe (post-quantum) alternative:

1. Signing

: ECDSA has significantly shorter signatures versus Lamport signatures.

1. Voting

: Fork-free voting with BLS signatures finalises significantly faster than forkful voting.

1. Random beacons

: Dfinity-style random beacons provide significantly better randomness versus blockhashes or a RANDAO approach.

1. Verification

: SNARK-based verification is significantly faster than execution-based verification.

The construction below uses cryptographic canaries and backups to simultaneously leverage the power of fancy primitives and the safety of safe primitives.

Construction

For every fancy primitive we want to use in-protocol we construct a canary contract with an associated large bounty. My guess is that 50,000 ETH is large enough. Such an amount can be subsidised by the protocol through inflation (0.05% inflation of the total supply is a small price to pay), or could be funded by donations from the community (I’d put 1 ETH

) and the foundation.

Anyone can redeem the bounty by producing a “proof of cryptographic threat”. The canary can be very specific, e.g. only target BLS signatures. In this case the proof could be a BLS signature matching a nothing-up-my-leave public key (e.g. the binary serialisation of "Come at me, bro"

). Alternatively the canary can be more general, e.g. target all pre-quantum crypto, where the proof would be a proof of quantum supremacy. A hash-based hiding commitment scheme is used (SHA3 is thought to be post-quantum secure).

We want the canary to be triggered before

any production crypto is at risk. For that the canary puzzle needs to be made easy enough that only the threat of a breakdown is displayed, not an actual complete breakdown. For example the puzzle in a post-quantum canary would be hard enough for no classical computer to stand a chance, but easy enough for a reasonably-low-qbit quantum computer to crack.

We now pair every fancy protocol-layer mechanism with a safe backup. The backup remains dormant until the canary is triggered. At that point all fancy operations are automatically and immediately shut down, and the safe backup takes over. This applies to system contracts (e.g. the VMC and the FFG contract) as well as to clients for offchain protocol rules (e.g. fork choice rules). Even application-layer contracts can listen to the canary and have their own contingency plan.