Joint work with the Coinbase cryptography team (and others)

- Arash Afshar
- Yi-Hsiu Chen
- Iftach Haitner
- Samuel Ranellucci

coinbase

Building MPC Wallets – Challenges and Solutions

Yehuda Lindell
Cryptographer

The Self-Custody Dilemma

Self Custody

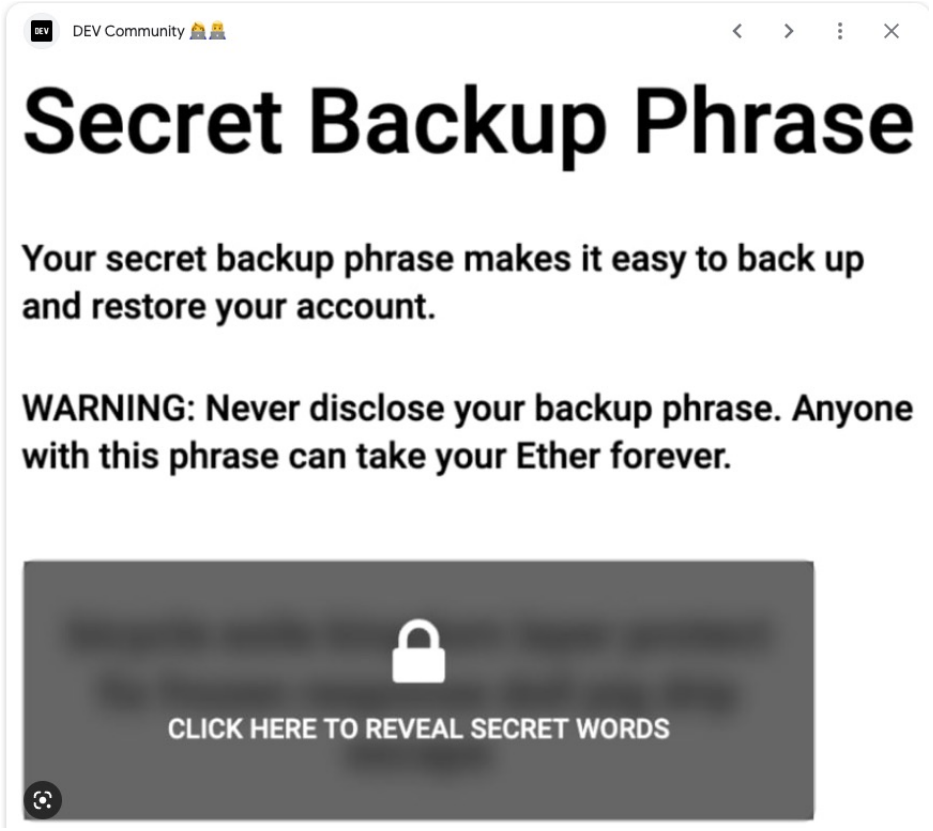
- **Technical solution**
  - User holds key – on mobile device, laptop, hardware device
- **Advantages**
  - Your keys your coins – not as a catchphrase: you have full control and this is the whole reason for decentralization
    - No one can censor you or prevent you using your keys (subpoena, other)
    - You are not reliant on a central organization, and you don’t lose your money if they go bankrupt
  - Other assets like NFTs are not aligned with a centralized exchange
    - Although it can be technically solved by them holding your keys for you

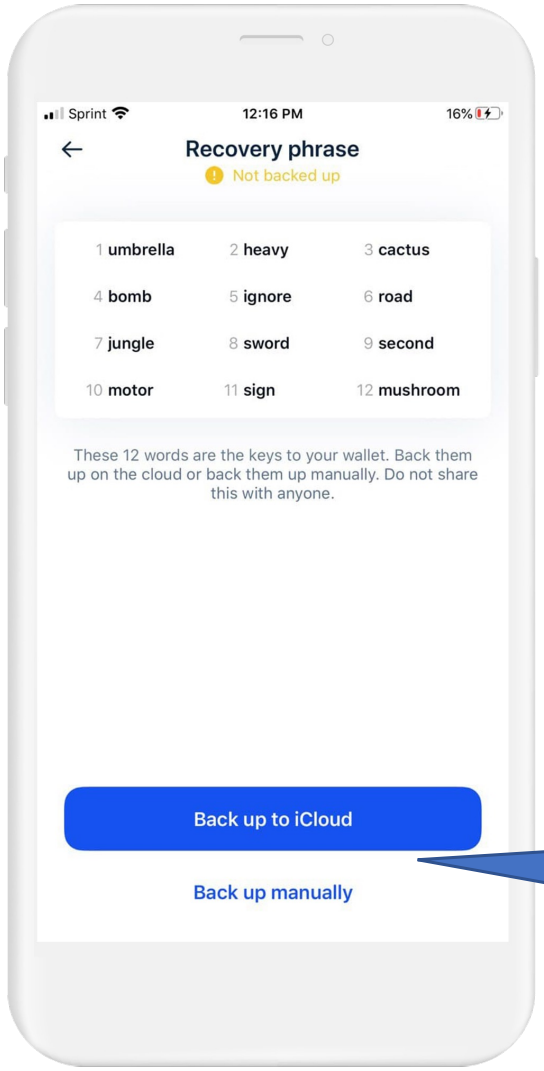
Self Custody

- **Technical solution**
  - User holds key – on mobile device, laptop, hardware device
- **Disadvantages**
  - Usability – you are responsible
    - Backup: store the mnemonic where it won't be **lost or stolen**
      - Note that these goals are at **direct odds** with each other
    - Many stories of users with wallets asking for password reset
  - Security
    - User devices are very problematic from a security perspective
    - Users are vulnerable to social engineering (exacerbated by mnemonic)
    - Backup is related here as well – consider backup in cloud storage, on a piece of paper, etc.

Self Custody Usability

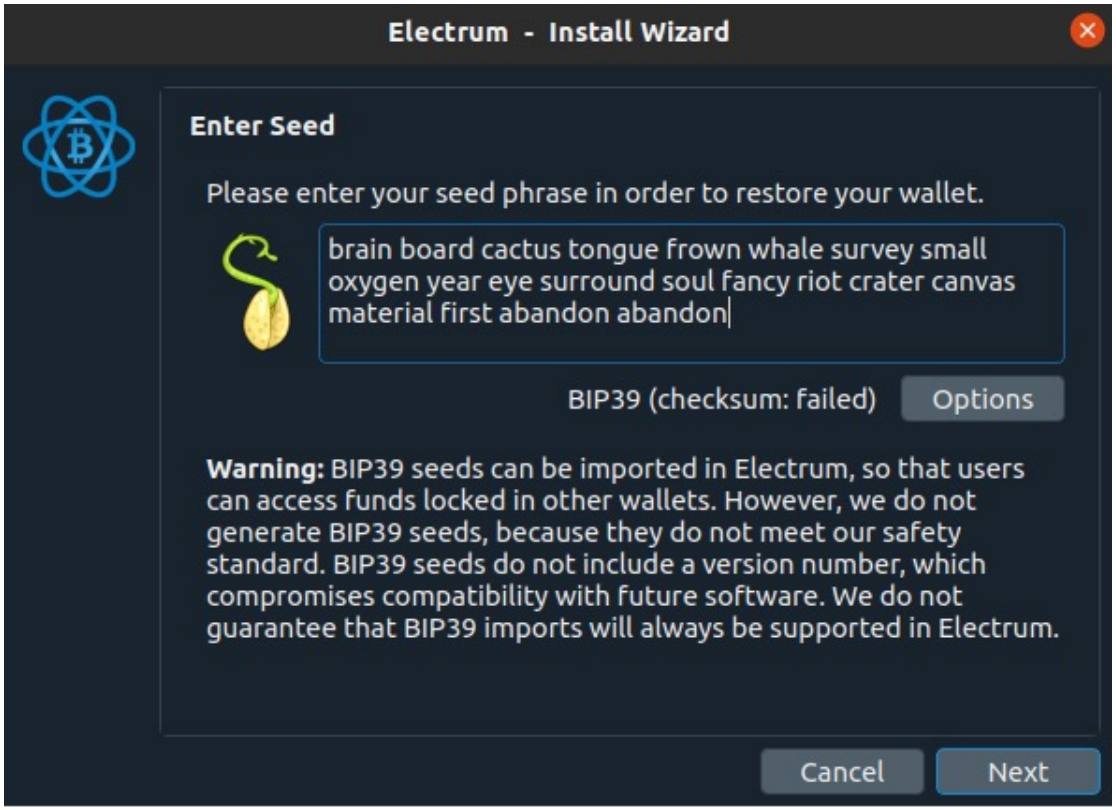
- **Think about the regular non-expert user**

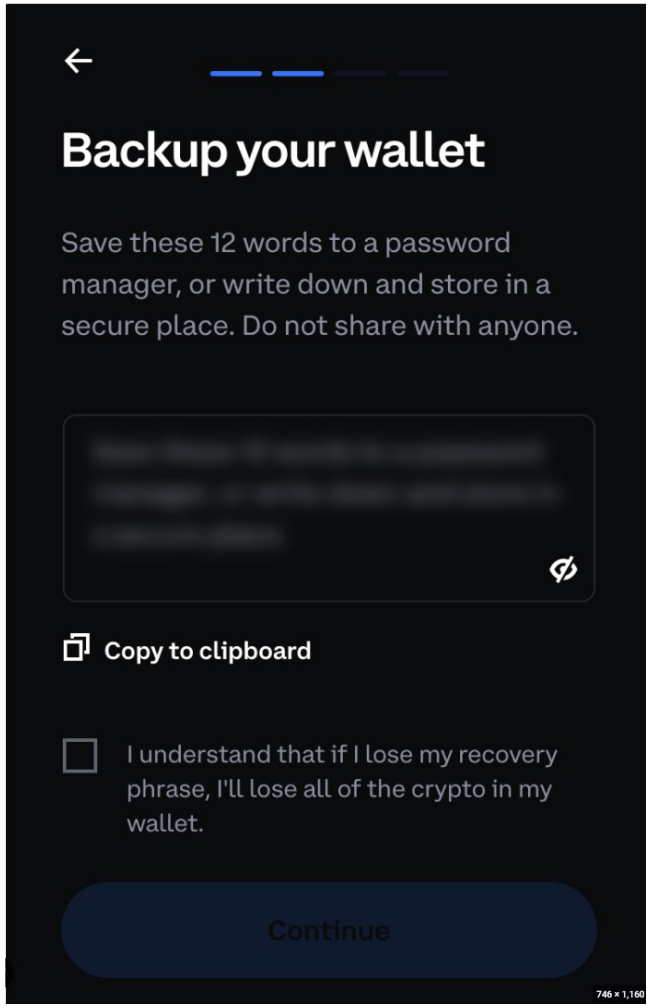




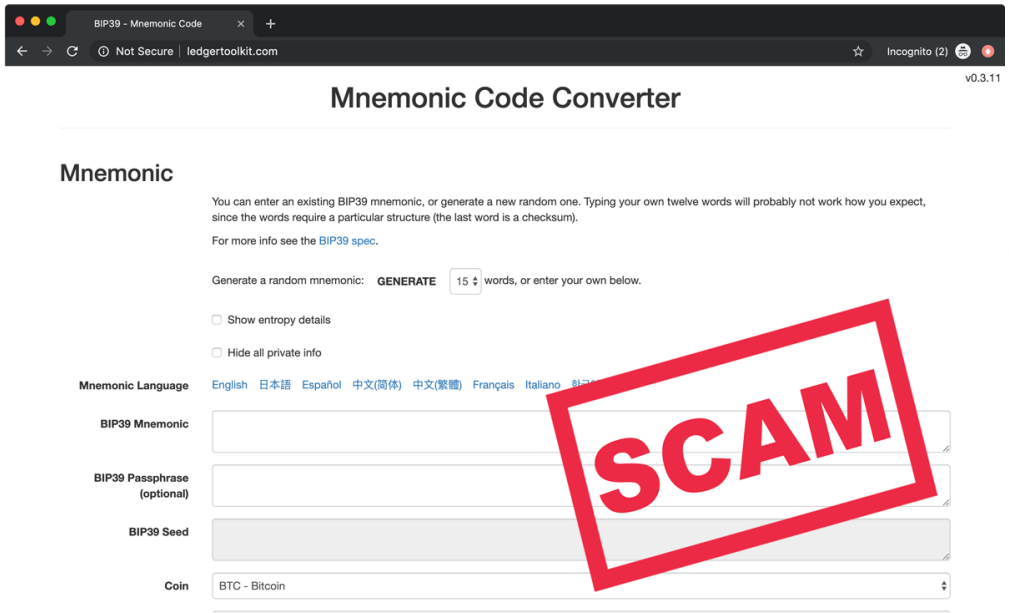
Self Custody Usability

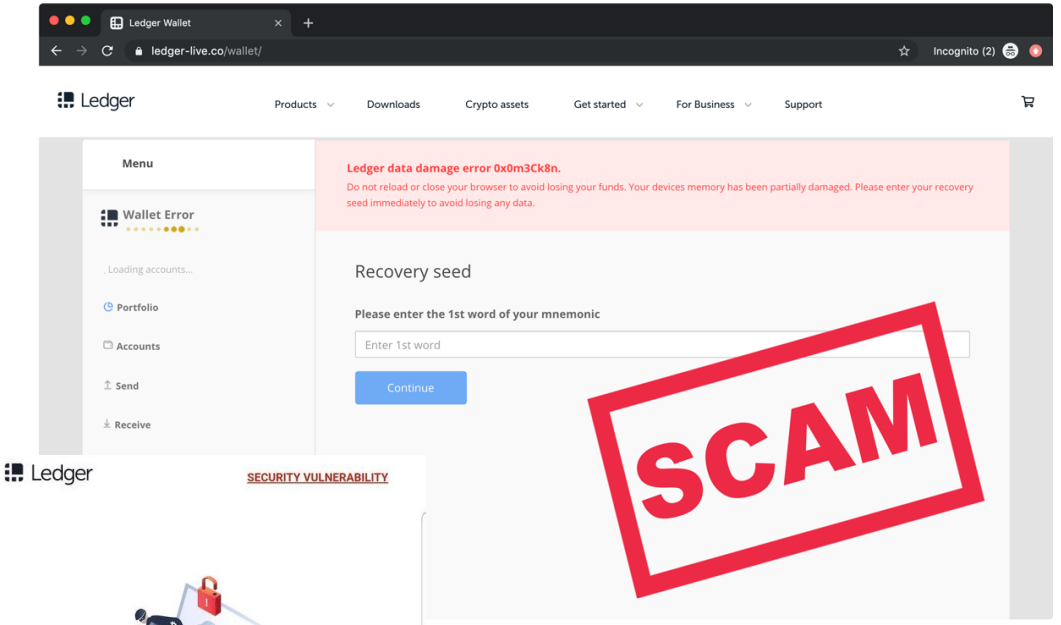
- **Think about the regular non-expert user**

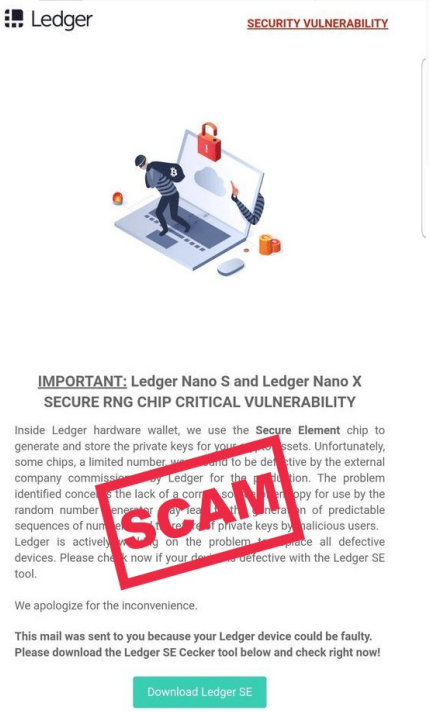




Self Custody Usability







Exchanges and Custody

- **Technical solution**
  - Exchange or custodian holds your funds for you
- **Advantages**
  - Burden of management and security is on them
  - Professional enterprises are far better at security and backup than regular users
  - Additional anti-fraud and other mechanisms can be used
    - Step-up authentication for high amounts, policies on amounts, allow-lists and more can be enforced at the exchange

Exchanges and Custody

- **Technical solution**
  - Exchange or custodian holds your funds for you
- **Disadvantages**
  - Not all exchanges are equal – users are vulnerable to bankruptcy or fraud by the exchange itself
    - True of regular banks, but we have decades of regulation and support to minimize these risks (at least in many countries)
  - Not decentralized so why bother (to some extent)

Exchanges and Custody

RFI

What next for French victims of the FTX cryptocurrency exchange collapse?

The collapse of the FTX cryptocurrency exchange has had repercussions around the globe, with more than a million clients losing money...

The Washington Post

Celsius bankruptcy judge ruling says account holders don't own their accounts

More than half a million people who deposited money with collapsed crypto lender Celsius Network have been dealt a major blow to their hopes...

CoinDesk

Three Arrows Capital Founders Launch Exchange Where You Can Trade 3AC Bankruptcy Claims

... Launch Exchange Where You Can Trade 3AC Bankruptcy Claims ... the founders of failed crypto hedge fund Three Arrows Capital (3AC),...

Exchanges and Custody

CRYPTOCURRENCIES

You Still Owe the IRS Even if Your Crypto Lender Collapsed

By Joe Light Follow Feb. 9, 2023 3:00 am ET Orde

Text size

Listen to article
5 minutes

Voyager Digital encouraged customers to get tax advice.
Dreamstime

Customers of failed crypto lenders such as Celsius Network, Voyager Digital , and BlockFi might have an unwelcome surprise this tax season.

Even though their earnings might be locked up in bankruptcy proceedings, the investors likely still owe taxes on much of what their accounts received last year.

Exchange and Custody

- **But – where would you put your money?**
  - In Sam's bank of the Bahamas, or
  - Citibank, JP Morgan,…
- **The same is true of crypto exchanges as well**

The Self-Custody Dilemma

- **If we want everyone (or anyone) to use cryptocurrency, then we must solve this problem**
- **The aim: self-custody with the experience of an exchange**

- **A note: I personally think that this isn't a XOR situation**
  - I want to keep some money in my wallet
  - I also want to keep some money in a "bank"
    - Bank in quotes due to regulatory ramifications

MPC Wallets – High Level Idea

Using MPC for Keys (aka Threshold Signing)

- **MPC considers "different parties" with "different inputs"**
- **We can also use MPC for one input split over different devices**
  - Take a private key k and "split" it into two random shares k_1 and k_2 such that $k_1 + k_2 = k$
  - Place one share on each device
  - Relate to each share as "private input" and run an MPC protocol
- **Security guarantees – a malicious adversary (running arbitrary attack code) having full control over one of the devices cannot break the protocol**
  - Privacy: attacker can't learn anything beyond the signature (so nothing is revealed about the key)
  - Correctness: an attacker can't make the signature be on a different transaction
  - Attacker needs to break into both devices in order to learn anything

Corruptions

- **Clearly, we aren't concerned with a corrupt user stealing from themselves**
- **We also aren't concerned with a corrupt service provider wanting to steal from their customers**
  - Of course, depending on the service provider
- **We are concerned that the user's device is infected by malware, that the service provider is breached or that there's a corrupt insider**
- **Key not at service provider also has major legal implications, but these alone can be solved with "semi-honest MPC"**

MPC-Based Wallets

- **Key is shared between user device (mobile and/or browser) and service provider**
- **Basic properties**
  - Service provider cannot transact without user (doesn’t hold key)
  - Malware on mobile isn’t enough for key theft
    - Key misuse is also mitigated with policies
  - Backup of user share is much easier (only one share)
  - **But**: naïve implementation is still not censorship-free
    - Need to add censorship-free backup

MPC Operations Needed

- **Signing (obviously)**
- **Key generation – never have a key exposed**
- **Refresh – force an attacker to "simultaneously" breach**
  - If shares of key are k_1, k_2 then update to $k_1 + r, k_2 - r$ for a randomly generated r
  - The sum is unchanged
- **Backup**
- **HD wallet support**

More About Backup

- **Publicly-verifiable backup**
  - The obvious idea: each device encrypts the share they generated under a (secured) backup key
  - The threat: one of the devices encrypts the wrong value
    - Why? Sabotage for example (competition, etc.)
  - Publicly-verifiable backup: verify that the encrypted value is correct without opening the encryption (or even holding the decryption key)

First Backup

- **Regular easy-to-use backup**
  - User holds private decryption key in cloud backup (as an example)
  - Service provider backs up their share locally
  - User backs up their share by encrypting under backup encryption key and sends to service provider
- **User loses their device, transfers and data is lost, etc.**
  - Service provider sends encrypted backup to user (strong authentication is needed here!)
  - User retrieves decryption key from cloud backup and decrypts
  - Service provider retrieves from backup (needed due to refresh)

Second Backup (if subpoena is a concern)

- **Censorship-free hard-to-use backup**
  - User holds private decryption key in secure environment
    - YubiHSM, biometric-protected secure enclave
  - Service provider encrypts their share under public backup key
  - User encrypts their share also under public backup key
  - Backup ciphertexts are stored (in device, in cloud, etc.)
  - Key is never exposed even while generating backup
- **User needs to export (is censored due to subpoena or anything else)**
  - User obtains both backup ciphertexts
  - User decrypts both shares

The Self Custody Dilemma

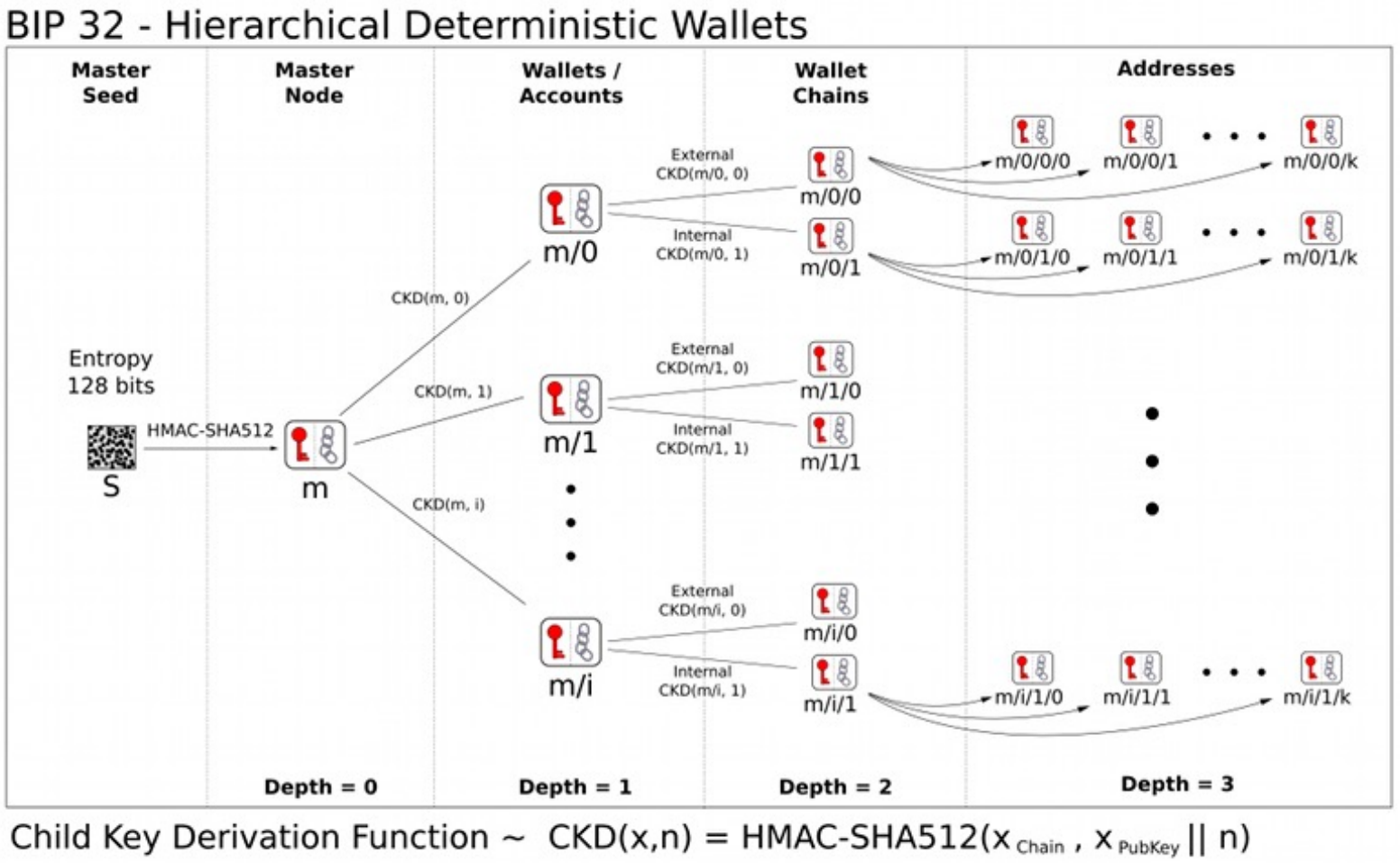
- **Why does this solve the mnemonic and usability problem?**
  - The backup is generated automatically and safely
    - Unlike with mnemonics, it is possible to store the user share in the cloud without too much danger (so it can be automatic)
    - The censorship-resistant backup is also generated safely
- **Other features**
  - Since the service provider is involved in all operations, they can apply policy and fraud mitigation mechanisms
  - Restore-from-backup provides a UX like password reset (as long as they have access to their cloud)

HD Wallet Support

- **Backup is “easier” with an HD wallet**
  - It isn’t essential since once backup is automatic, we can do it for every generated key, but it requires access to backup storage which depends on the setting

- **Can we build MPC HD wallets?**

BIP-039/BIP-032 Compliant HD Wallets



BIP-039/BIP-032 Compliant HD Wallets

- **Why do we want BIP compliance?**
  - Standard methods require no explanation
  - Existing BIP wallets can be imported
    - It’s possible to import just hard-derived keys, but this is actually not so simple
      - What happens if I run the wallet in parallel in two different products?
      - What if I import to an MPC wallet that doesn’t yet support everything I have?
  - Export is much easier with fully compliant methods
    - Just export a mnemonic or seed (not the same thing)
    - No technical problem with exporting keys (and this can be done) but it depends on the support in the other wallet

Construction BIP-Compliant Wallets

- **MPC theorem: any probabilistic polynomial-time function can be securely computed**
  - Convert function to Boolean circuit and compute
  - But how efficient is it?
- **BIP-039**
  - A way of generating a seed from a mnemonic
  - Uses PBKDF2: 2048 iterations of HMAC-SHA512 [Yehuda's rant]
  - Size of single garbled circuit (essentially a lower bound here):
    - Approximate number of AND gates in circuit: $4 \times 58{,}120 \times 2048 = 476{,}119{,}040$
    - Size of circuit = 32 bytes per AND gate; approx 14GB
  - This is not happening anytime soon from a mobile…

Constructing BIP-032 Compliant Wallets

- **Run two-party MPC of HMAC-SHA512**
  - Three HMAC-SHA512 for a derivation (BIP-044)

- **But isn't two-party malicious secure computation very expensive?**

HD Derivation – It's Not So Simple

- **MPC ensures that the computation is correct; it says nothing about the inputs (think about trusted black box)**
  - What forces the parties to input the correct shares of the seed in the derivation?
  - What forces the parties to use the correct output shares from the derivation?
    - Computing the public key in the circuit would add tens of millions of gates
- **Input and output enforcement mechanisms need to be added**

Constructing BIP-Compliant Wallets

- **Imported wallet**
  - Only have mnemonic, and anyway it was already in one place, so do local key generation and then split
  - Any later hardened derivations are via MPC on the seed with BIP-032
- **New wallet – fully BIP-039 compliant**
  - Can do the same as above (still much better than standard wallets)
- **New wallet – BIP-032 compliant**
  - Can generate the seed and run MPC to generate all keys
  - Expensive but possible
- **New wallet – not BIP compliant; can export keys only**

MPC-Friendly HD Wallets

- **Use an MPC-friendly derivation function**
  - This means that it is amenable to efficient MPC
  - Typically functions with nice algebraic structure

- **Same problems of input and output enforcement exist**
  - Not considered by most existing solutions (as far as what I've seen)

Summary

- **MPC wallets can solve some of the major problems of usability**
  - Easier backup since only one share
  - Achieve effect of “password reset”
  - Enable policy enforcement, fraud management, etc.
- **MPC solutions required:**
  - Standard key generation, signing, refresh etc.
  - Publicly-verifiable backup
  - HD wallet derivation with input/output enforcement
  - MPC-friendly derivation
  - And more, like deterministic signing, EdDSA key-compatibility,…

Thank You

coinbase

Building MPC Wallets – Technical Details

Yehuda Lindell
Cryptographer

Technical Details

- **Publicly-verifiable backup**
- **BIP-032 derivation in MPC**
  - Computation
  - Input and output enforcement
- **MPC-friendly HD derivation**
- **Deterministic signing**
  - Why is this needed?
- **EdDSA key-compatibility**
  - What's the problem? Isn't EdDSA just Schnorr (ignoring nonce generation)?

Publicly-Verifiable Backup

- **Task: given private x and public $Q = x \cdot G$, and given public encryption key pk, generate C so that**
  - Given (C, Q, pk) an efficiently verify that C encrypts the dlog of Q
- **This is quite easy using additively homomorphic encryption**
- **We want to use any encryption scheme (RSA, ECIES, etc.)**
  - Enables storage of backup keys in HSMs, smartcards, secure enclaves, or anywhere
- **Note: when doing distributed key generation, each party will backup their share of the private key**

Interactive Proof with Soundness ½

- **Input:** (x, Q, pk)
- **Prover commit:**
  - Choose random x_0, x_1 such that $x_0 + x_1 = x$
  - Compute $Q_0 = x_0 \cdot G$ and $Q_1 = x_1 \cdot G$
  - Compute $C_0 = Enc_{pk}(x_0; r_0)$ and $C_1 = Enc_{pk}(x_1; r_1)$
  - Sends (Q_0, C_0, Q_1, C_1)
- **Verifier challenge: send random challenge b to open first or second**
- **Prover response: send (x_b, r_b)**
- **Verifier: check that**
  - $C_b = Enc_{pk}(x_b; r_b)$
  - $Q_b = x_b \cdot G$
  - $Q = Q_0 + Q_1$

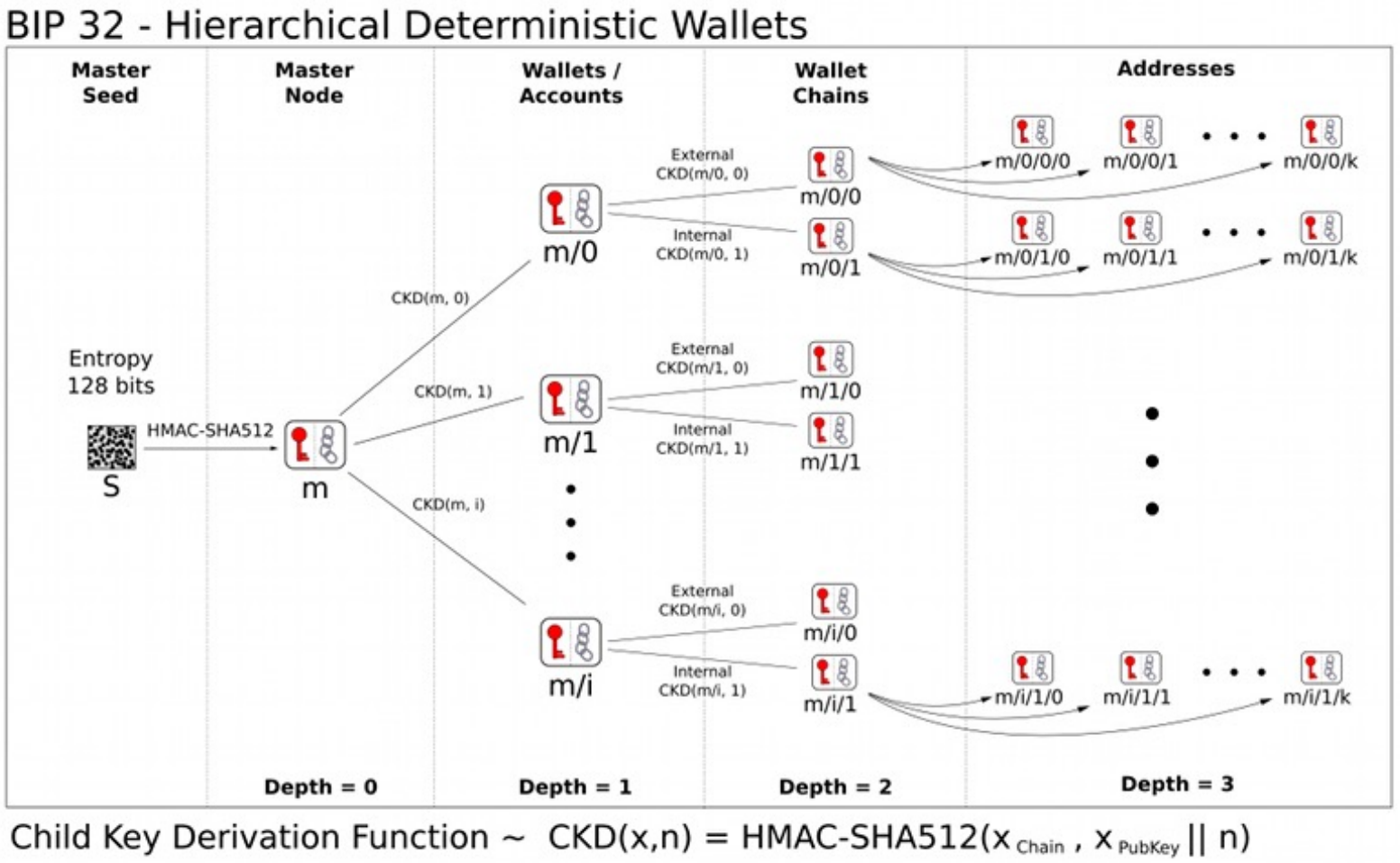
Soundness: if both checks would pass, then this implies that the encryptions sum to dlog of Q

Zero-knowledge: revealing only one ciphertext gives nothing (just random garbage); simulate by computing (x_b, r_b, C_b, Q_b) and take $Q_{1-b} = Q - Q_b$

Non-Interactive Proof

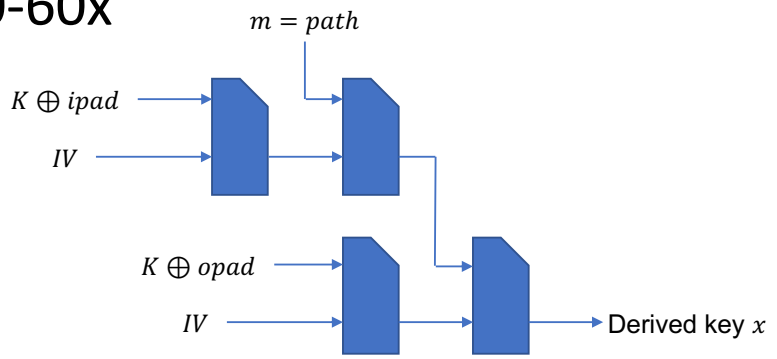
- **Needed for public verifiability**
- **Solution:**
  - Run in parallel 128 times and use Fiat-Shamir
  - Optimizations
    - Provide randomness and not ciphertext
    - Reduce bandwidth (and increase work) by building tree

BIP-032 Compliant HD Wallets



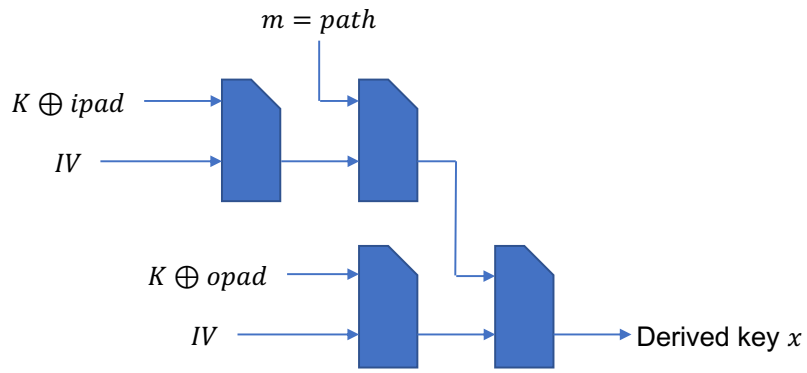
Constructing BIP-032 Compliant Wallets

- **Naïve: run fully malicious 2PC on derivation circuit per key**
  - Each derivation requires 3 HMAC-SHA512 computations:
    - Each HMAC is four SHA computations
    - Size: $4 \times 58{,}120 \times 3 = 687{,}440$ AND gates (size of a single garbled circuit = 21.3MB
  - Fully malicious protocols
    - Garbled-circuit cut-and-choose: about 40-60x
    - Authenticated garbling: about 10x
  - Can be too expensive



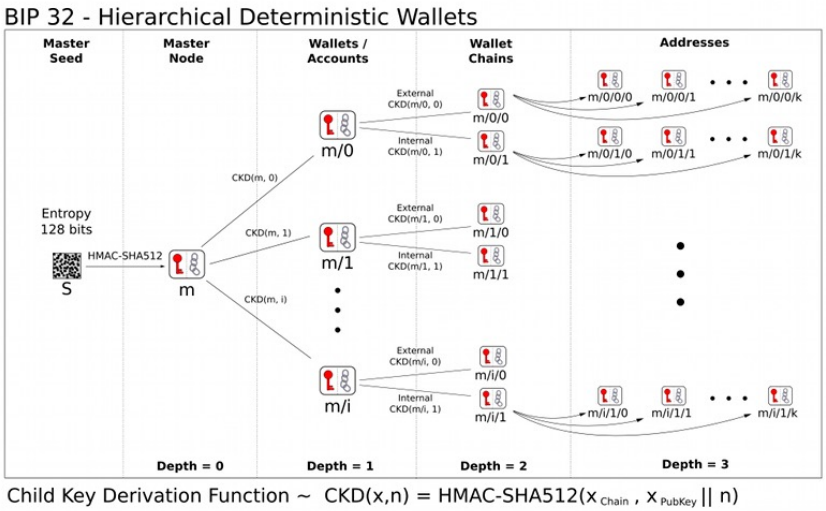
Stage 1 – Smaller Circuit

- **Sometimes big improvements come from small observations**
  - In BIP-032, the HMAC key is the (non or semi-private) chain code
    - If we provide the chain code to both MPC parties, then can reduce HMAC from 4 to 2 SHA512 computations
    - This requires breaking the circuit computation into 3 parts and forcing correct (private) output to be used



Stage 1 – Smaller Circuit

- **Sometimes big improvements come from small observations**
  - Further improvement by keeping intermediate values in tree



Child Key Derivation Function ~ CKD(x,n) = HMAC-SHA512(x_{Chain} , x_{PubKey} || n)

Stage 2 – Malicious Two-Party Computation

- **Dual execution**
  - Basic garbled circuit construction is secure against malicious evaluator (if OT is malicious secure)
    - OT + garbled circuit
  - Challenge for achieving malicious security
    - Garbled circuit can be incorrect and can even reveal secrets
  - The dual execution method
    - Run basic construction in both directions (evaluator gets output only)
    - Compare results and only reveal output if the same

Stage 2 – Malicious Two-Party Computation

- **Dual execution is leaky**
  - Malicious circuit: “if first bit input is 0 then output correct; else output garbage”
  - The party learns the bit from whether or not there’s an abort
- **Dual execution isn’t sufficient for many standard MPC tasks**
  - Can reveal “most important bit”
- **Dual execution for key derivation**
  - Input is random and learning a bit is OK
  - **Challenges**
    - If can run many in parallel, then can learn entire key
    - Party can always abort if it’s about to be caught in equality check, and claim “my machine fell”

Dual Execution – Distinguishing Aborts

- **Aim: ensure recovery from accidental aborts**
- **Solution:**
  - In equality check, encrypt recovery information (small)
    - Use encryption key in backup
  - If equality check doesn't conclude, then set flag to recover
  - After recovery check, reset flag
  - Check flag before any execution
- **And make sure no parallelization**

Output Enforcement – The Problem

- **We run dual execution to obtain private shares x_1, x_2 of derived key**
- **At this point, we can run distributed key generation using these values (instead of choosing random shares)**
- **But:**
  - What forces the parties to actually input x_1, x_2?
  - If they don't, backup will be invalid!

Output Enforcement – Naïve Solution

- **Naïve solution**
  - Garbled circuit computes $Q = x \cdot G$ and outputs it
  - After key generation, compare to given Q
- **Problem**
  - This requires millions of gates!

Output Enforcement – Solution

- **Parties also input random a_1, b_1 and a_2, b_2**
  - Circuit computes derived key x (shares x_1, x_2)
  - Circuit also outputs $t = (a_1 + a_2) \cdot x + (b_1 + b_2)$
    - Prefer computation over integers; can set a of 64 bits, and b of size 64 bigger than $a \cdot x$ (concretely 384)
- **After execution:**
  - Party 1 sends commitment to $Q_1 = x_1 \cdot G$ and $B_1 = b_1 \cdot G$ and a_1
  - Party 2 sends $Q_2 = x_2 \cdot G$ and $B_2 = b_2 \cdot G$ and a_2
  - Party 1 decommits
  - Both check that $t \cdot G = (a_1 + a_2) \cdot (Q_1 + Q_2) + (B_1 + B_2)$
  - If yes, they output Q

Output Enforcement – Solution

- **Recall**
  - $t = (a_1 + a_2) \cdot x + (b_1 + b_2)$
  - Both check that $t \cdot G = (a_1 + a_2) \cdot (Q_1 + Q_2) + (B_1 + B_2)$
- **Soundness**
  - Parties reveal their values before seeing the others
  - If Party 1 wants to change Q_1 to Q_1' then it needs to find a_1', B_1' such that
    - $(a_1 + a_2) \cdot (x_1 + x_2) + (b_1 + b_2) = (a_1' + a_2) \cdot (x_1' + x_2) + (b_1' + b_2)$
    - $\Leftrightarrow a \cdot x + b = (a + \Delta_a) \cdot (x + \Delta_x) + (b + \Delta_b)$
    - $\Leftrightarrow 0 = a \cdot \Delta_x + \Delta_a \cdot x + \Delta_a \cdot \Delta_x + \Delta_b$
  - But a is not known
- **Privacy: b is large enough to hide a, given t**

Input Enforcement

- **But parties can input different seed in different computations**
  - In particular, can input different seed than what was backed up
- **Solution**
  - Generate seed and back it up
    - Using publicly-verifiable backup, we know a public Q_{in} for the seed
  - Run a circuit computation with input seed shares
    - Can use the same method of output enforcement on Q_{in} as well
    - Can also compute one-time MAC for future executions

MPC-Friendly HD Wallets

- **Use an MPC-friendly derivation function**
  - This means that it is amenable to efficient MPC
  - Based on functions with nice algebraic structure
    - Strongly recommend against circuit-efficient hash
- **We will give an imperfect yet reasonable solution here**
- **Tool – VRF (party committed to PRF)**
  - For example:
    - In setup, provide $K = k \cdot G$; H is random oracle to curve
    - $Y = PRF_k(m) = k \cdot H(m)$ (by DDH, this looks random)
    - Proof that $(G, H(m), K, Y)$ is a Diffie-Hellman tuple (easy proof)

Imperfect MPC-Friendly Derivation

- **Hold shares of a root key x_1, x_2 with public key $Q = (x_1 + x_2) \cdot G$**
- **Each party uses a VRF (with a different key) to derive some Δ_1, Δ_2 from the path, and set $\Delta = \Delta_1 + \Delta_2$**
- **The derived public key is $Q_{new} = Q + \Delta \cdot G$**
- **The private key shares are $x_1 + \Delta$ and x_2**
- **This is like normal derivation, except only MPC parties can compute it**
  - Doesn't support delegation or export of some keys (typically OK)
  - Unlinkability is still supported for anyone except MPC participants (but they can link anyway)

Deterministic Signing – why do we care?

- **Maybe you are concerned about randomness generation**
  - Personally I'm less concerned in an MPC setting, but…
- **Sometimes you have no choice**
  - How do some dApps utilize wallets that can only do ECDSA/EdDSA but they want to do other things?
    - Upon enrollment, ask for two signatures on a random message
    - If they are the same, then support the wallet; else reject
    - Derive the key by asking for a signature on a fixed message
    - Interesting fact: $H(Sign_{sk}(m))$ is a PRF

Deterministic Signing – Danger

- **Naïve solution: each party uses PRF to locally derive randomness**
- **Attack**
  - Attacker uses different randomness on two signatures
  - Schnorr
    - $R = r \cdot G; \quad s = r + H(m||R) \cdot k$
      - $r = r_1 + r_2$ chosen by the parties
    - Attacker sets $r_i' = r_i + \Delta$ for a known Δ
    - Given $s = r + H(m||R) \cdot k$ and $s' = r + \Delta + H(m||R) \cdot k$, compute $k = \frac{s - s' - \Delta}{H(m||R)}$

EdDSA

- **Key setup**
  - SHA512 hash key k to 512 bits
    - First part: signing private key x; signing public key $Q = x \cdot G$
    - Second part: derivation key d
- **Sign message m**
  - Nonce: $r = SHA512(d, m)$ and $R = r \cdot G$
  - $e = SHA512(R, Q, m)$
  - $s = r + e \cdot x \mod q$
  - Output (R, s)
- **EdDSA is Schnorr with key and randomness derivation**
  - Schnorr is very MPC friendly

EdDSA Compatibility

- **If we don't care about deterministic signatures, we can simply use any MPC protocol for Schnorr**
  - This is indistinguishable (up to signing twice on same message)
- **In many (or most) cases this is "good enough"**
- **What about compatibility?**
  - No problem: export generated Schnorr key as EdDSA-derived x
    - This will work for all EdDSA implementations (take d random)
    - Regeneration of signatures won't work – not sure it matters

EdDSA Export Compatibility

- **But *all* wallets expect to receive the pre-derivation key k**
  - This is true also of wallets that enable key import (and not just BIP import)
- **This means that I need to generate shares of x from shares of k, so that it can be exported**
  - Or I could provide my own proprietary code, but…
- **Solution:**
  - Everything we saw with BIP derivation and enforcement, but just one SHA512 computation

Summary

- **MPC wallets solve a lot of problems and are a great fit**
- **But naïve solutions can be very dangerous**
- **There are many advanced MPC techniques needed here, and lots of research questions for improving them**

Thank You