vetKeys: How a blockchain can keep many secrets

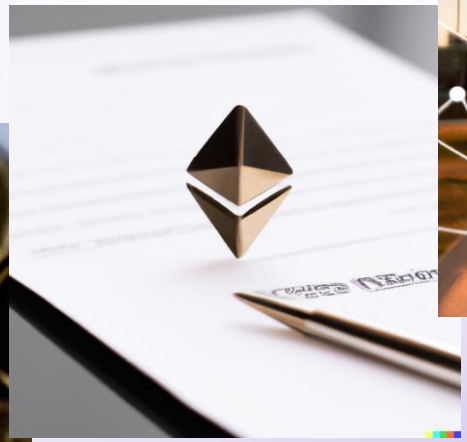
Andrea Cerulli[1], Aisling Connolly[1], **Gregory Neven**[2], Franz-Stefan Preiss[1], Victor Shoup[1]

[1] DFINITY

[2] Previously DFINITY, soon Chainlink Labs

Blockchain evolution

payments



smart contracts

Web3: decentralized applications (dapps)
e.g., DeFi, SocialFi, GameFi

Confidentiality on blockchains

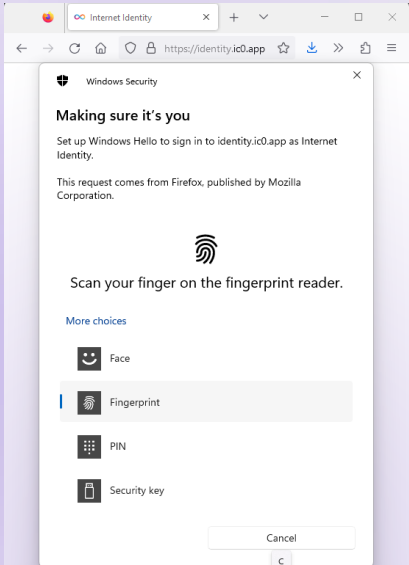
“Secure by default”

- ☑ Integrity
- ☑ Availability
- ☑ Censorship resistance
- ✘ Confidentiality

Actually worse than Web2:
cleartext data on many servers or even on public blockchain

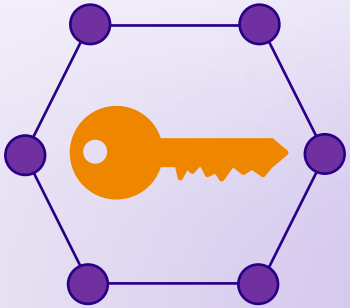
How to improve confidentiality on Web3

- Trusted execution platforms (Intel SGX, AMD SEV-SNP)
  poor track record against adversaries with physical access
- Secret sharing: doesn’t scale
- ZK/MPC/FHE: works for limited functionality, less for general execution e.g., payments (Zcash, Monero)
- Encryption key management by users
  - Lacking hardware support
    (hardware wallets, trusted elements)
  - Browser storage: insecure & inconvenient
  - Poor user experience syncing devices



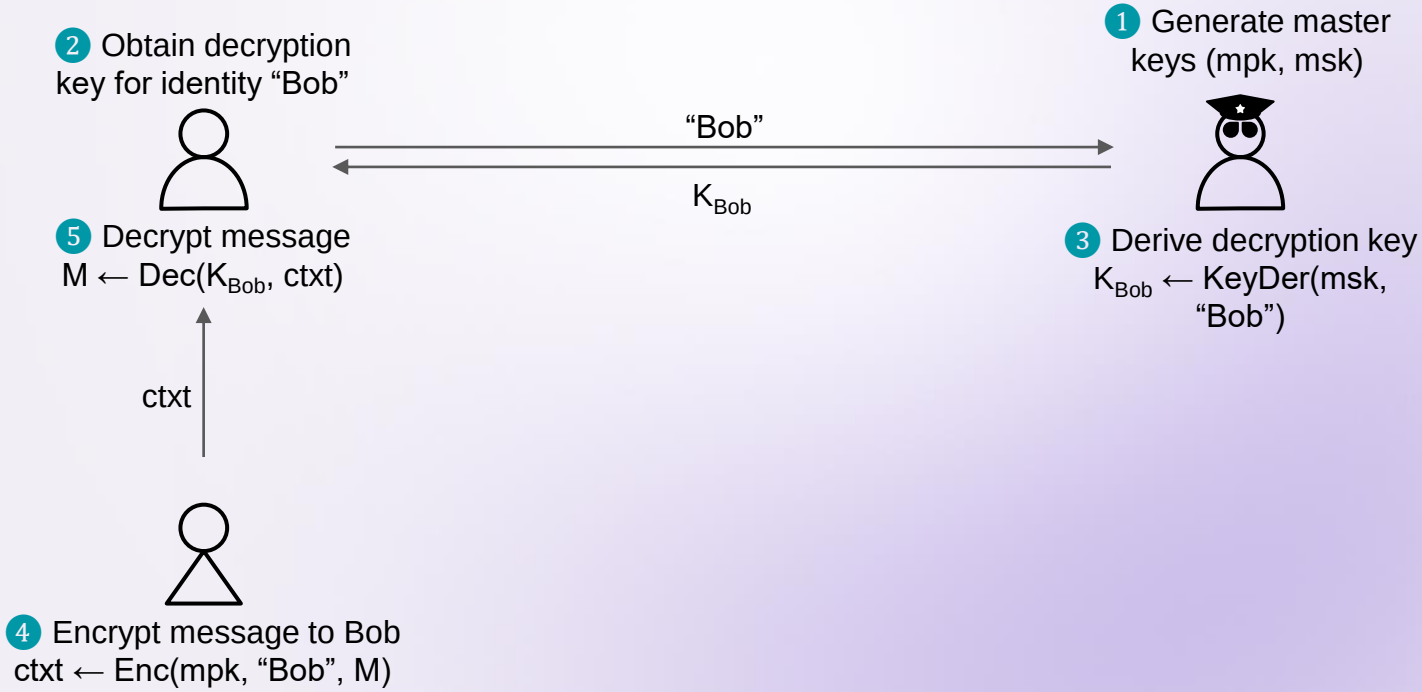
Verifiably encrypted threshold keys (vetKeys)

- Single master key, secret-shared among nodes
- Dapps **deterministically** derive **strong cryptographic keys** for symmetric/asymmetric encryption, signatures,…
- Derived keys can be delivered to dapp, or straight to user

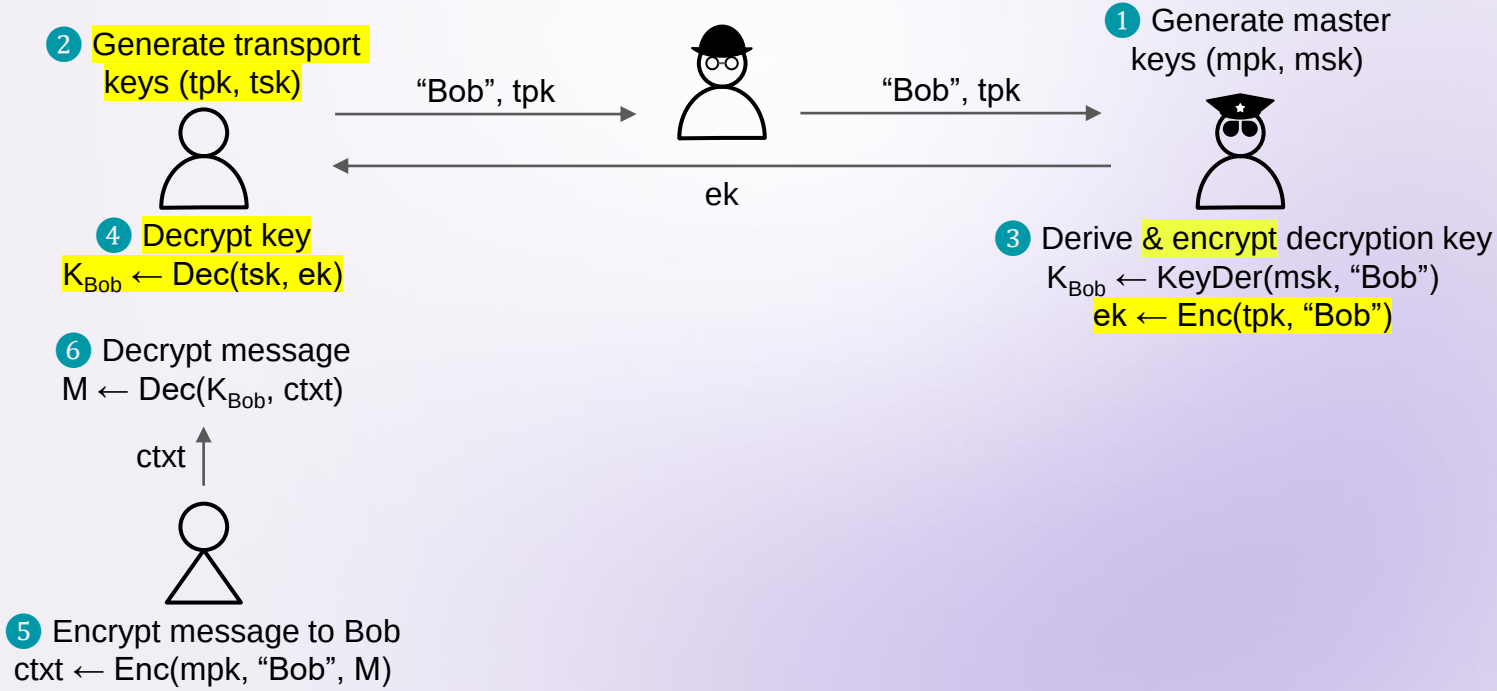


Concept

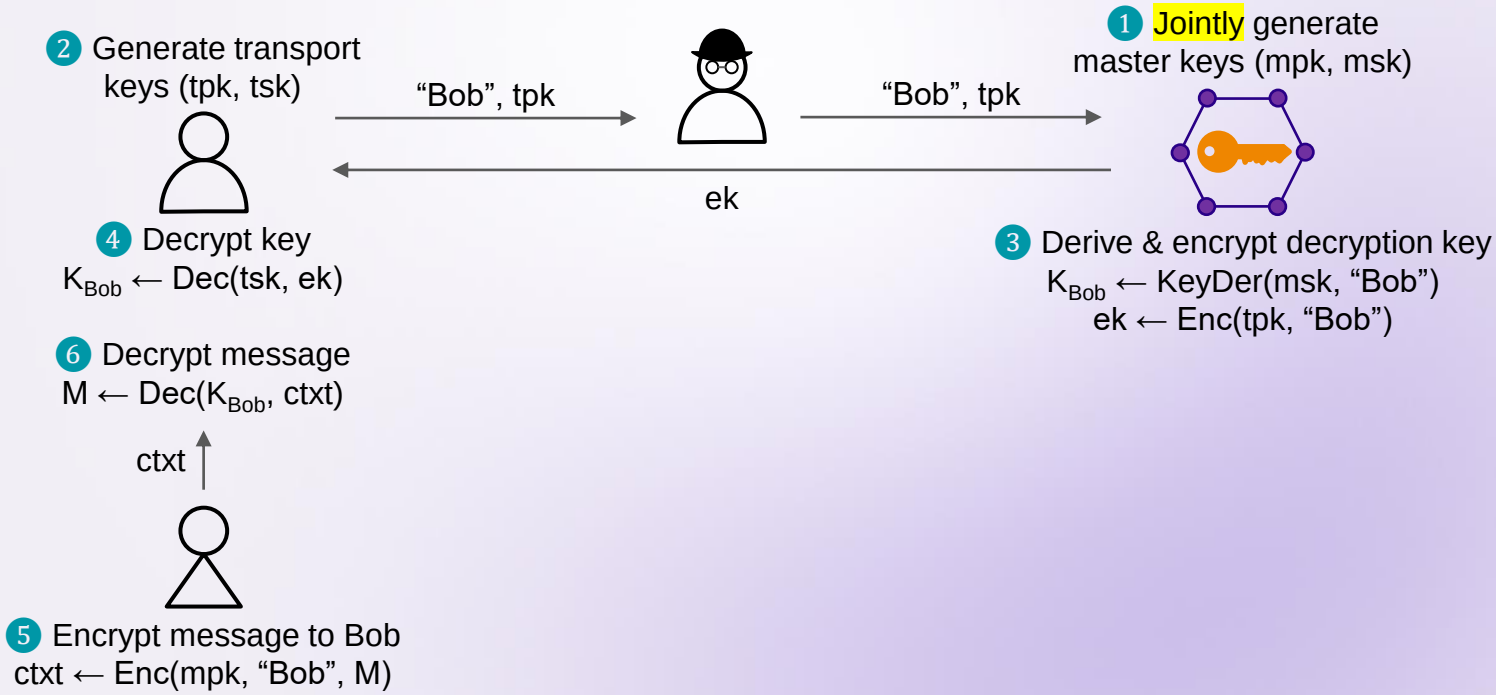
Identity-based encryption (IBE)



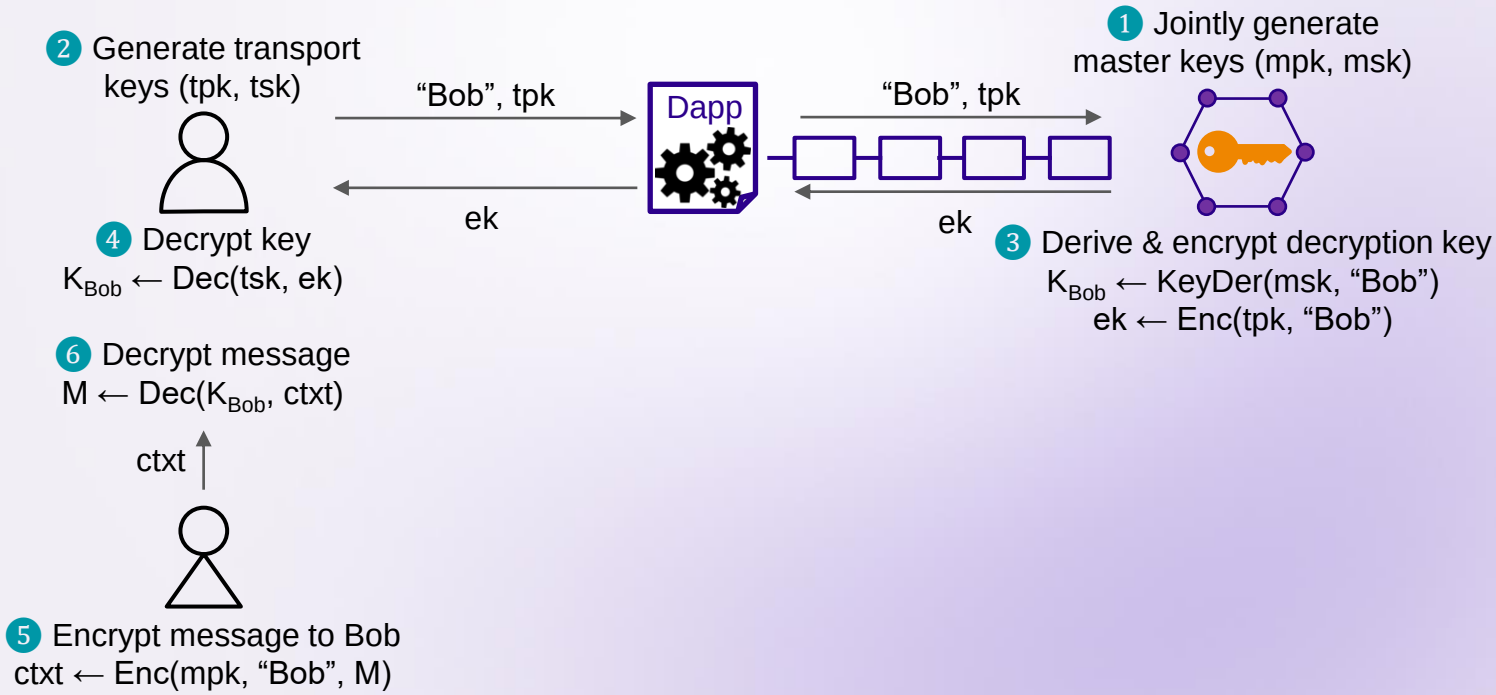
Identity-based encryption (IBE)



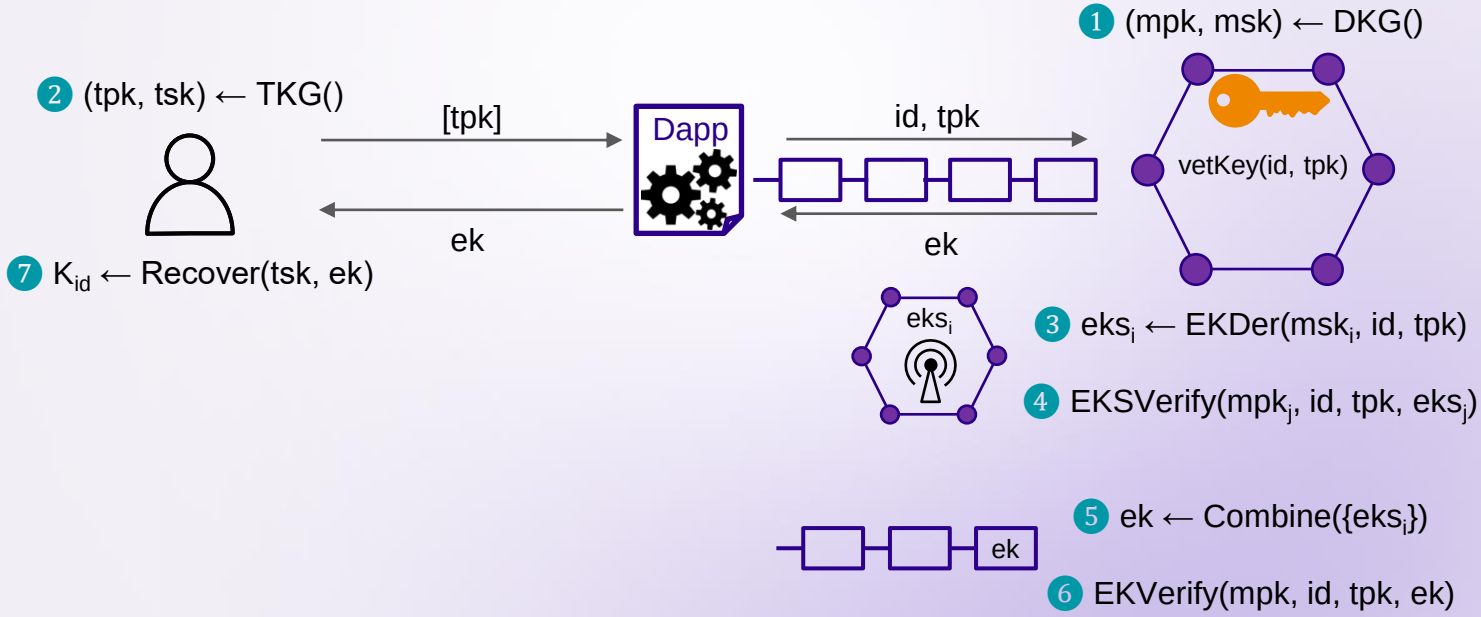
Identity-based encryption (IBE)



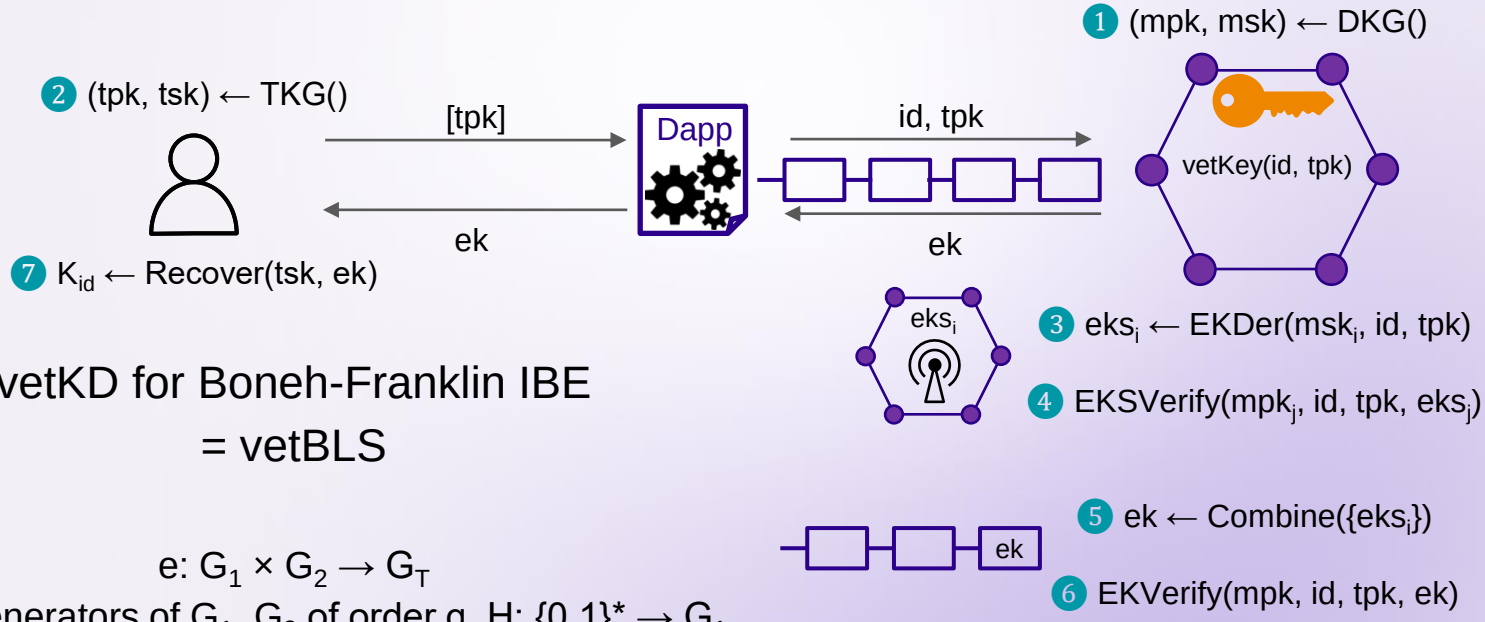
Identity-based encryption (IBE)



Verifiably encrypted threshold key derivation (vetKD)



Verifiably encrypted threshold BLS



vetKD for Boneh-Franklin IBE
= vetBLS

$e: G_1 \times G_2 \rightarrow G_T$

g_1, g_2 generators of G_1, G_2 of order q, $H: \{0,1\}^* \rightarrow G_1$

$pk = g_2^{sk}$

Sign: $\sigma = H(m)^{sk}$

Verify: $e(\sigma, g_2) = e(H(m), pk)$

Verifiably encrypted threshold BLS

1 (mpk, msk) ← DKG()

2 (tpk, tsk) ← TKG()

vetKey(id, tpk)

3 eks_i ← EKDer(msk_i, id, tpk)

4 EKSVerify(mpk_j, id, tpk, eks_j)

5 ek ← Combine({eks_i})

6 EKVerify(mpk, id, tpk, ek)

7 K_{id} ← Recover(tsk, ek)

vetKD for Boneh-Fra
= vetBLS

e: $G_1 \times G_2 \rightarrow G$

g_1, g_2 generators of G_1, G_2 of ord

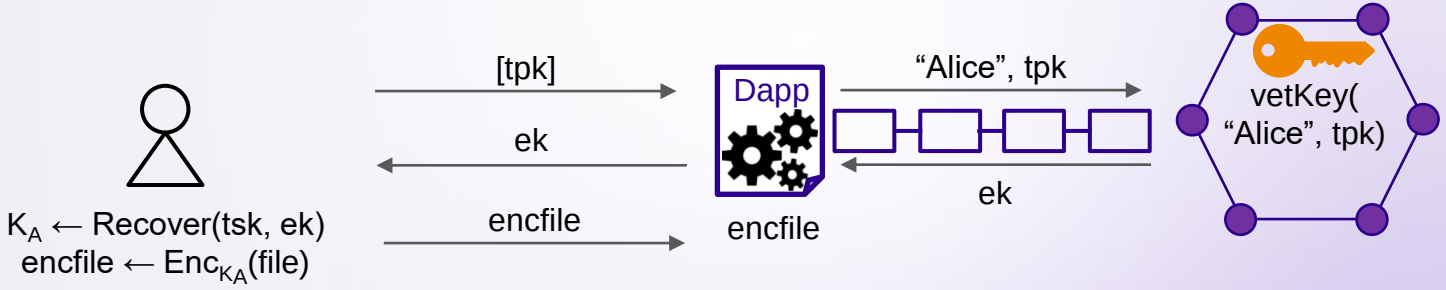
$pk = g_2^{sk}$

Sign: $\sigma = H(m)$

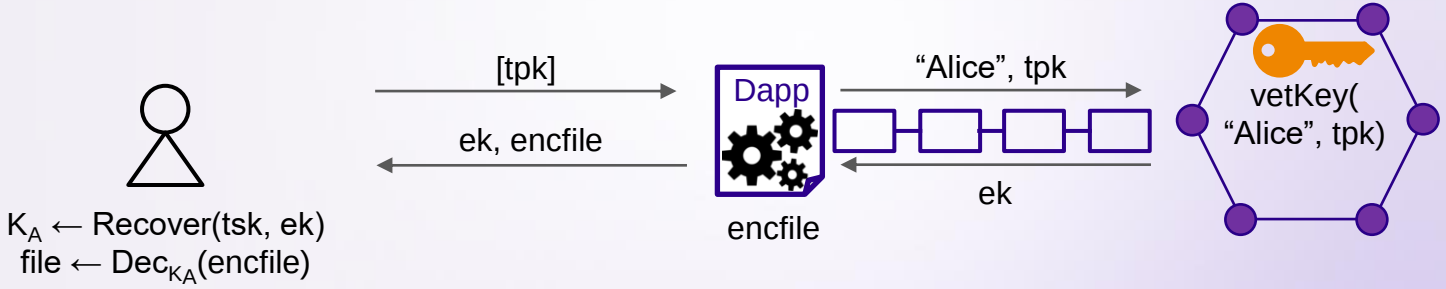
Verify: $e(\sigma, g_2) = e(H$

Use cases

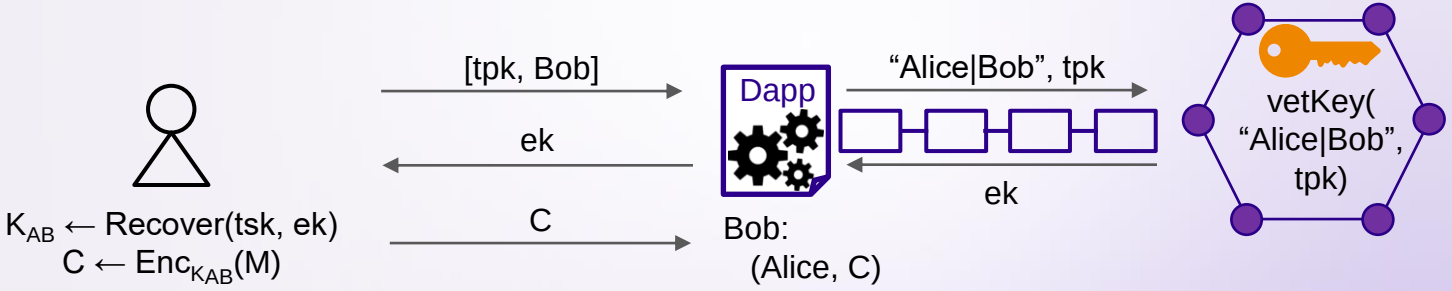
End-to-end encrypted storage



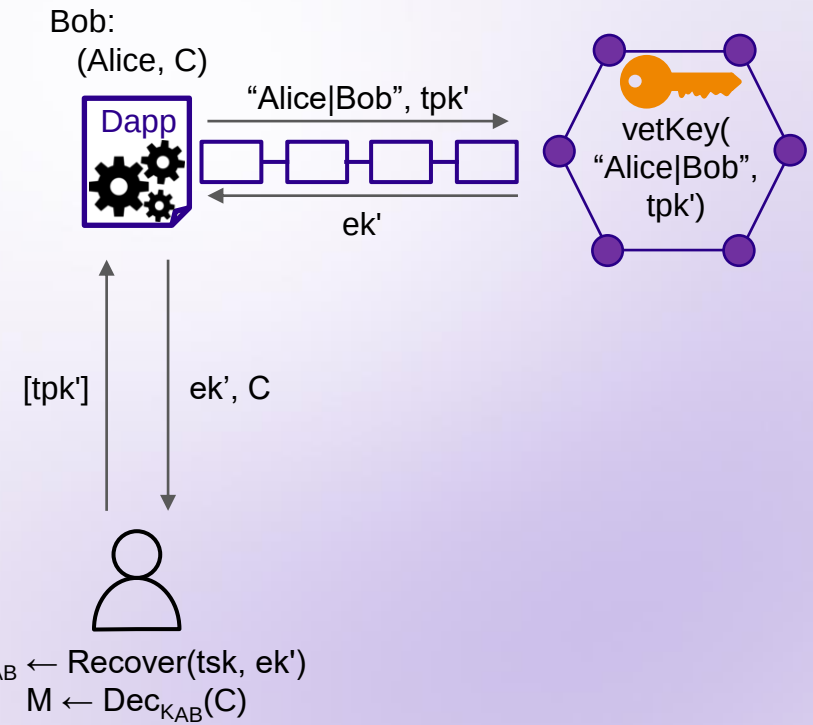
End-to-end encrypted storage



End-to-end encrypted messaging

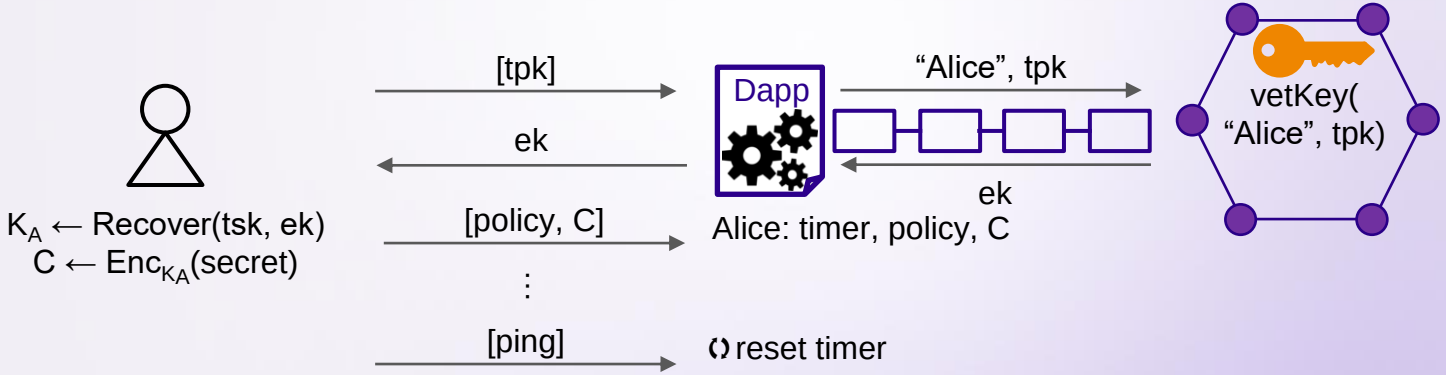


End-to-end encrypted messaging



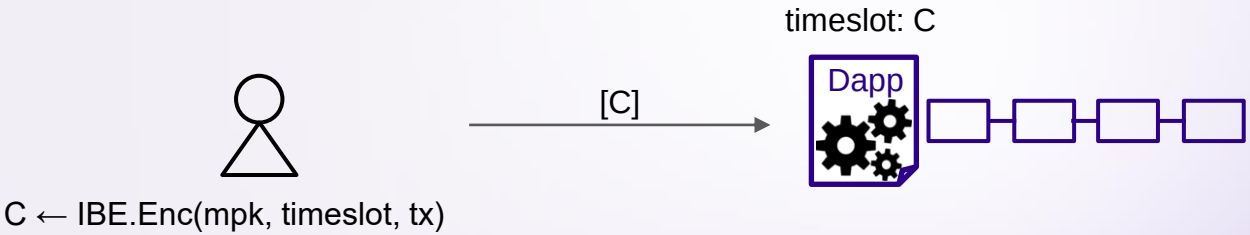
Similarly: decentralized E2EE social networks, letting authorized users derive vetKey(postid)

Dead man’s switch



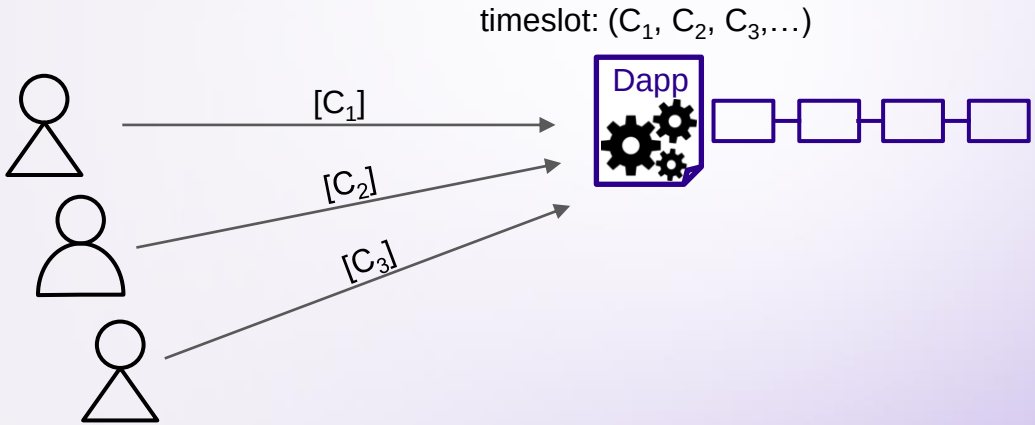
- Journalists, dissidents, whistleblowers: policy reveals secret to accomplice or publish (by revealing vetKey to Dapp itself using “dummy” tpk)
- Digital inheritance: policy as “digital will”, e.g., passwords, crypto(-currency) key

Prevent miner-extracted value (MEV)



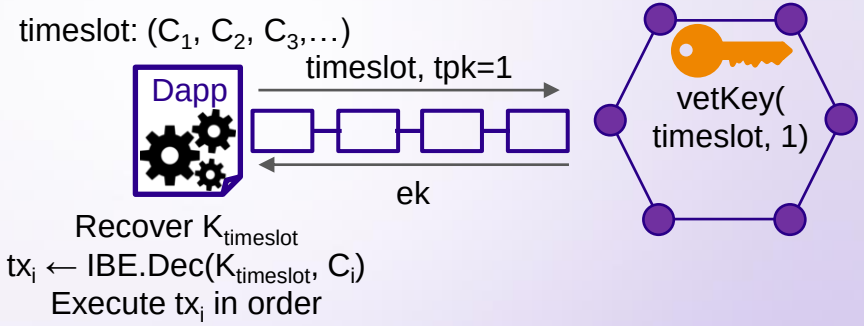
Front-running on digital exchanges (DEX):
more than $600M already lost to MEV since 2020

Prevent miner-extracted value (MEV)



Front-running on digital exchanges (DEX):
more than $600M already lost to MEV since 2020

Prevent miner-extracted value (MEV)



Front-running on digital exchanges (DEX):
more than $600M already lost to MEV since 2020

Similarly: secret-bid auctions, deriving vetKey for auction id

Other vetKey use cases

- **Time-lock encryption:** encrypt messages to the future
  see next talk!
- **Witness encryption:** encrypt to $x \in L$, decrypt using witness w
  Dapp derives vetKey(x, tpk) only if user provides [w, tpk]
- **One-time programs:** can only be executed on a single input
  garbled circuit using vetKey(wire|b) as wire encoding,
  Dapp only derives one vetKey per wire
- **Cross-chain bridging:** e.g., swap assets across blockchains for DeFi
  use vetKey as signature by Dapp, instead of implementing light clients

… and all of that with a single secret-shared key!

Other vetKey use cases

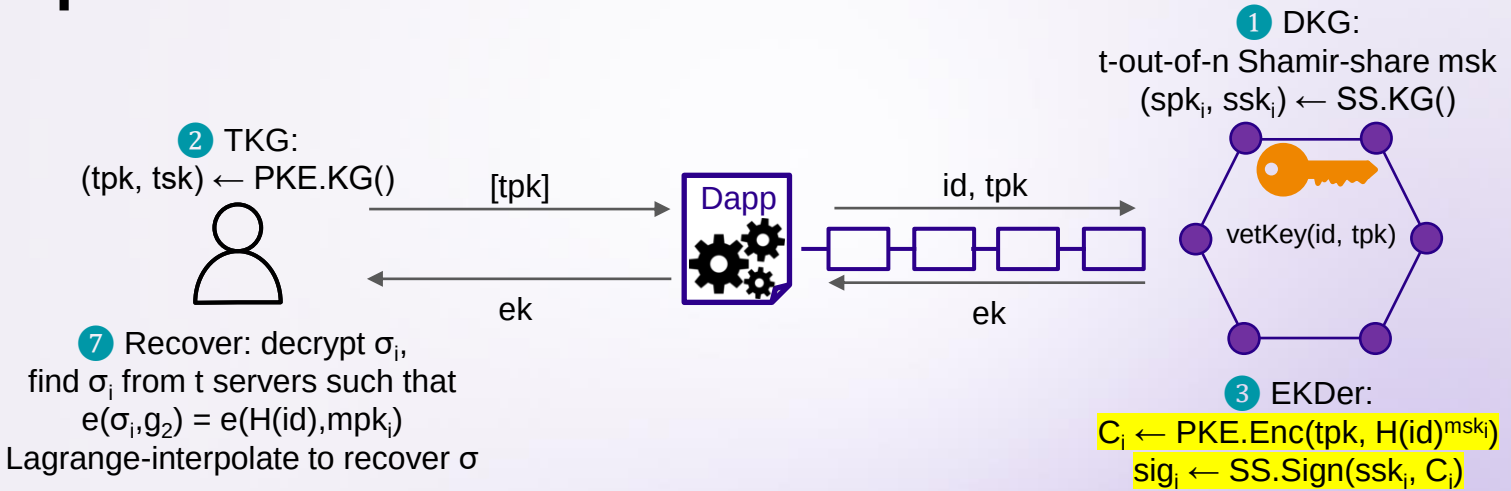
- **Time-lock encryption:** encrypt messages to the future
  see next talk!
- **Witness encryption:** encrypt to $x \in L$, decrypt using witness w
  Dapp derives vetKey(x, tpk) only if user provides [w, tpk]
- **One-time programs:** can only be executed on a single input
  garbled circuit using vetKey(wire|b) as wire encoding,
  Dapp only derives one vetKey per wire
- **Cross-chain bridging:** e.g., swap assets across blockchains for DeFi
  use vetKey as signature by Dapp, instead of implementing light clients

Previously only with trusted hardware or indistinguishability obfuscation (iO)
(status: complicated)

… and all of that with a single secret-shared key!

Schemes

A simple vetBLS scheme



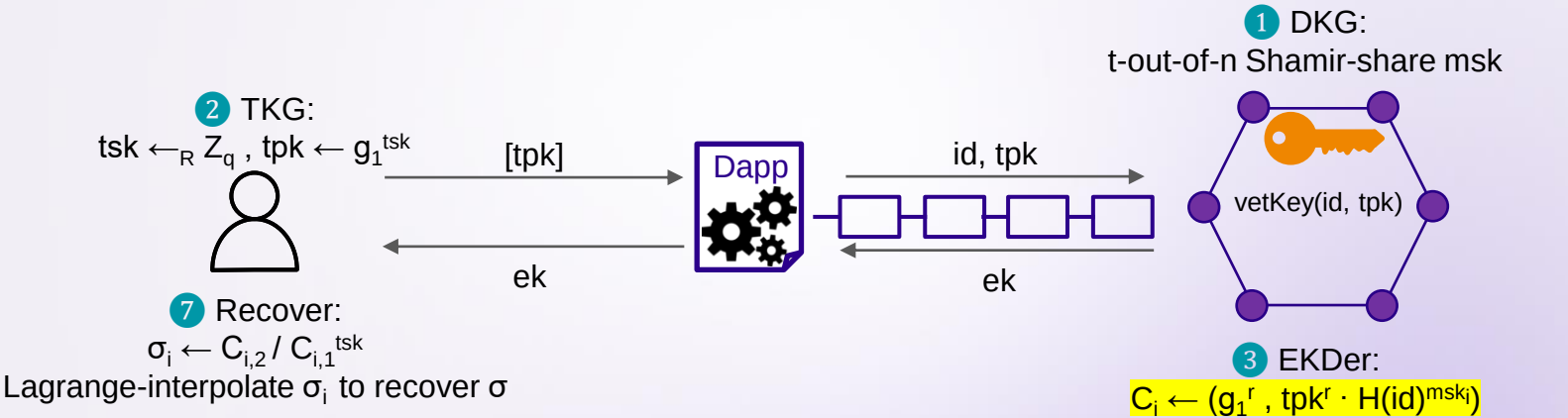
Encrypt and sign σ_i with standard PKE & SIG

ek = 2t-1 ciphertexts with sigs

≥ t of which are valid when decrypted

$t < n/2$

A zero-knowledge vetBLS scheme



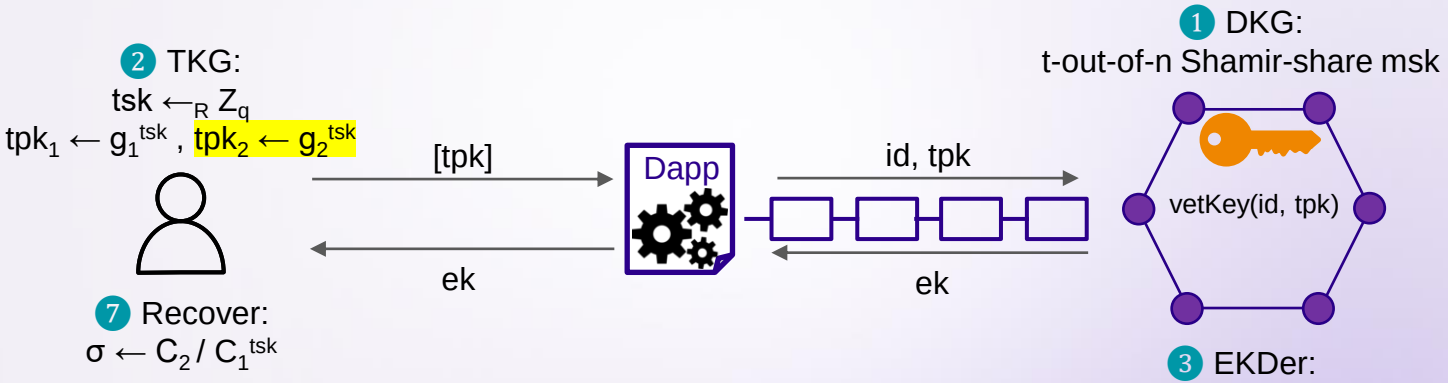
ElGamal-encrypt σ_i in G_1 (XDH assumption)
add zero-knowledge proof of correct encryption
concatenate t ciphertexts with proofs

4 EKSVerify: verify π_i

5 Combine: $ek \leftarrow (C_i, \pi_i)_{i \in S}$
for $|S| = t$

6 EKVerify: verify all π_i

An aggregatable vetBLS scheme (1)



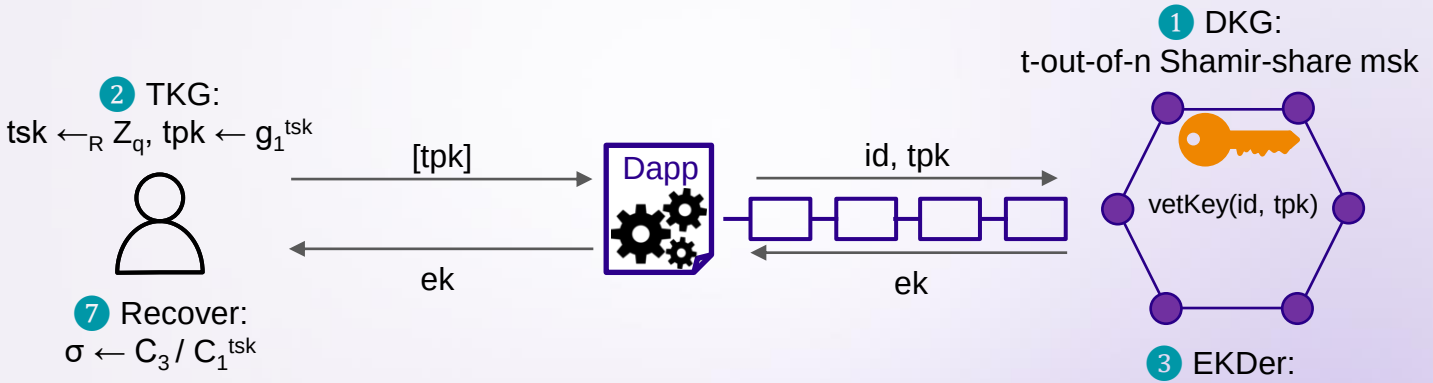
ElGamal-encrypt σ_i in G_1, verify using $tpk_2 = g_2^{tsk}$
(leaks info, but fine for IBE/PRF/VRF/SIG)
compact ek by interpolating ciphertexts
threshold variant of VES in [BGLS04]

4 EKSVerify: check
$e(C_{i,2},g_2) = e(C_{i,1},tpk_2) \cdot e(H(id),mpk_i)$

5 Combine: Lagrange-interpolate
$(C_{i,1},C_{i,2})$ to (C_1,C_2)

6 EKVerify: check
$e(C_2,g_2) = e(C_1,tpk_2) \cdot e(H(id),mpk)$

An aggregatable vetBLS scheme (2)



4 EKSVerify: check
$e(C_{i,1},g_2) = e(g_1,C_{i,2})$ and
$e(C_{i,3},g_2) = e(tpk,C_{i,2}) \cdot e(H(id),mpk_i)$

5 Combine: Lagrange-interpolate
$(C_{i,1},C_{i,2},C_{i,3})$ to (C_1,C_2,C_3)

6 EKVerify: check
$e(C_1,g_2) = e(g_1,C_2)$ and
$e(C_3,g_2) = e(tpk,C_2) \cdot e(H(id),mpk)$

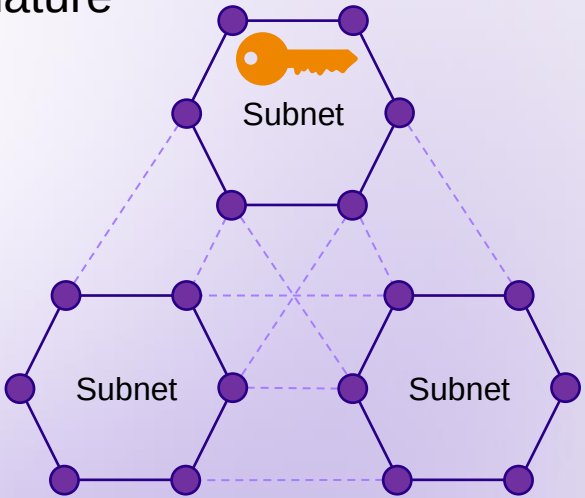
ElGamal-encrypt σ_i in G_1, verify using $C_2 = g_2^r$
(leaks info, but fine for IBE/PRF/VRF/SIG)
compact ek by interpolating ciphertexts
threshold variant of VES in [BGLS04]

Integration into the Internet Computer

The Internet Computer

internetcomputer.org

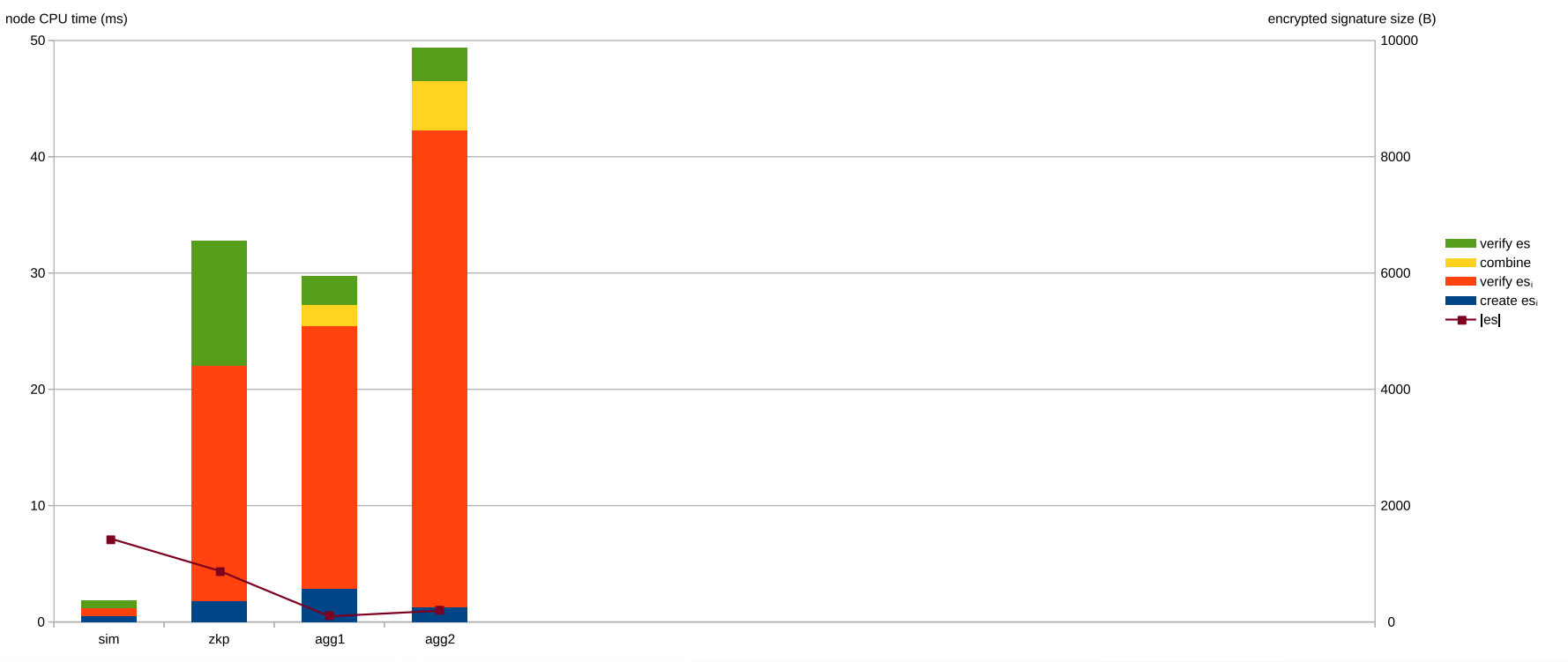
- Each subnet has own blockchain and threshold signature key for state certification and cross-net messaging
- Fast enough to host certified web dapps, including certified front-end
  → **crypto in Javascript/WASM makes sense!**



- Plan: install vet master keys on some subnets, keep backup on others
- Domain separation between dapps by prefixing dapp id

Performance

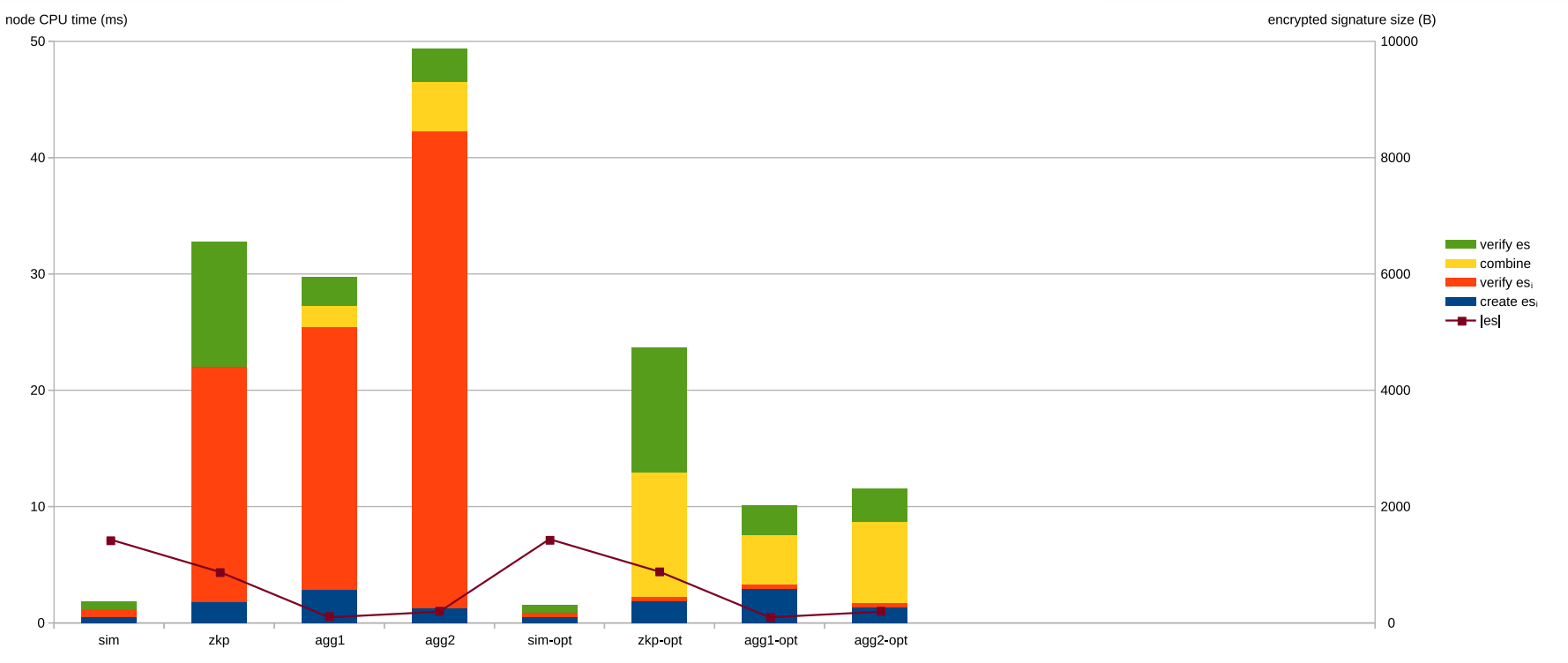
Performance estimates



Throughput: 20-500 vetKeys/s per server core (5 out of 13)

7-225 vetKeys/s per server core (14 out of 40)

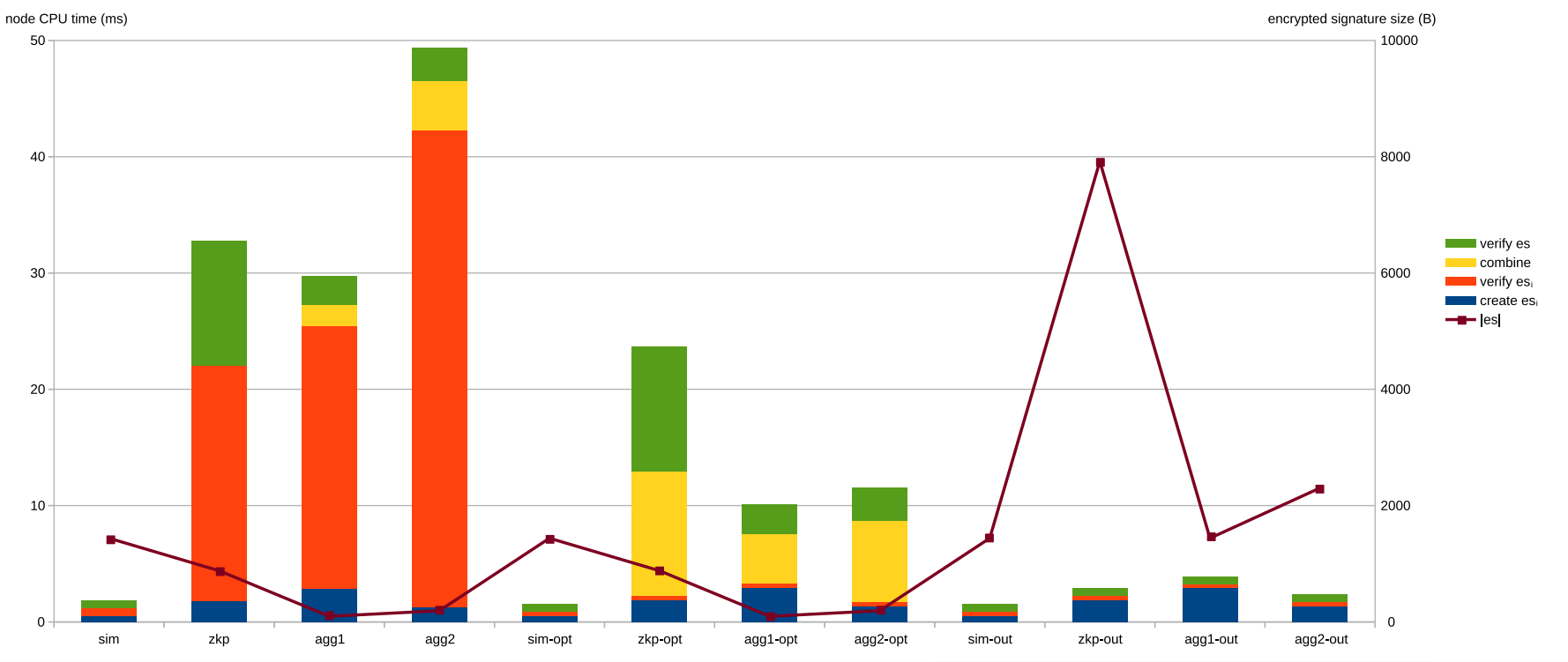
Performance estimates



Optimistic verification: skip share verification, combine and verify

Throughput: 80-640 vetKeys/s per server core (5 out of 13)

Performance estimates



Outsourced verification: sign and concatenate, let user verify ($t < n/2$)

Throughput: 420-650 vetKeys/s per core (5 out of 13)

Conclusion

vetKeys for privacy on Web3

- Research paper landing on ePrint soon, implementation in the works
- Many applications: E2EE, MEV prevention, encrypt to future events
  Only limit is dapp developers' creativity.
- Efficient: single master key, throughput of ~100 vetKeys/s
- Bonus feature: partially blind signatures from 2nd aggregated scheme

  Untraceable token transfers, anonymous credentials, oblivious transfer, OPRF-like password strengthening, private inputs to one-time programs