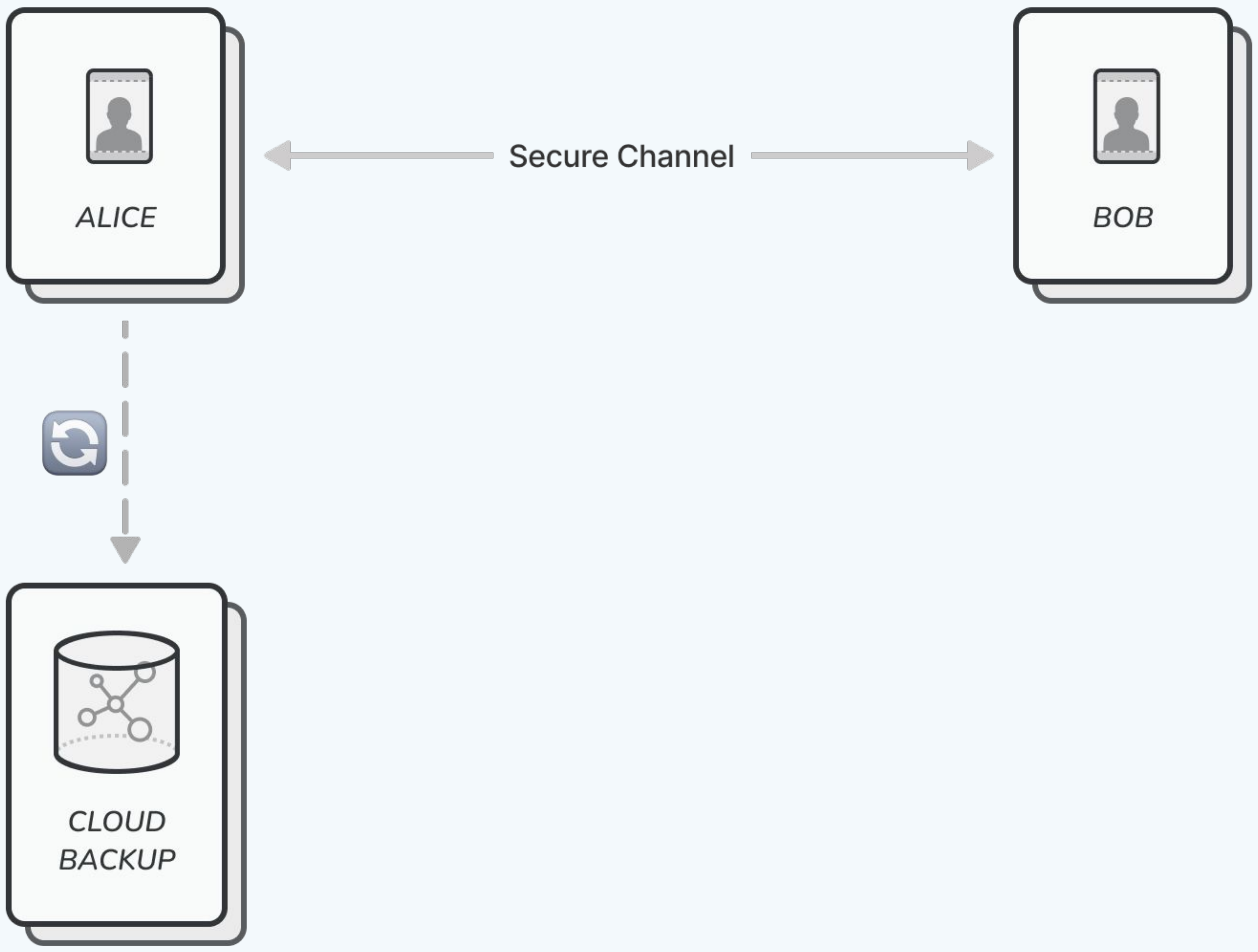
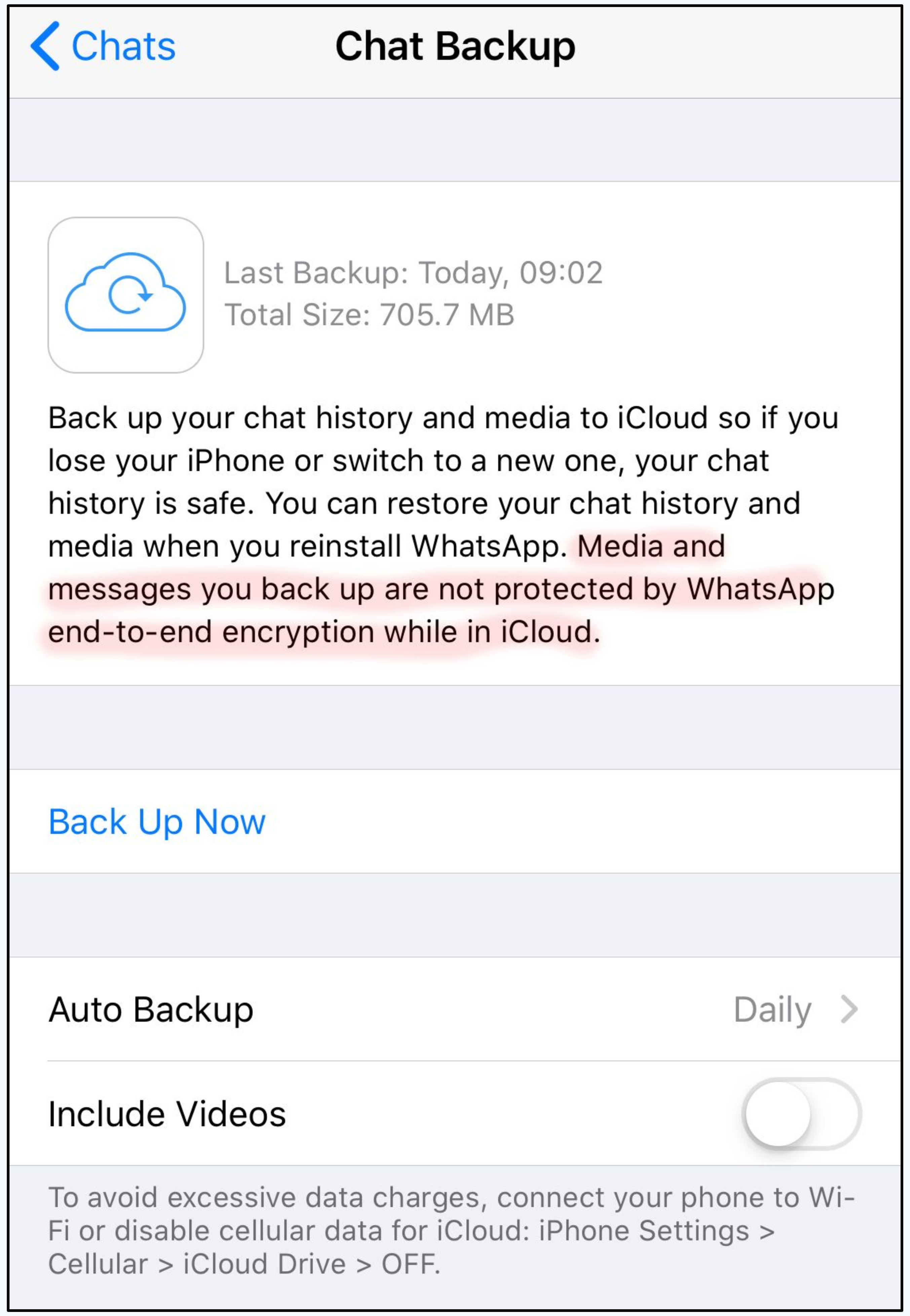
WhatsApp End-to-End Encrypted Backups

Kevin Lewi

(Unencrypted) Cloud Backups





End-to-End Encrypted (E2EE) Backups

Goal: Full privacy of message content

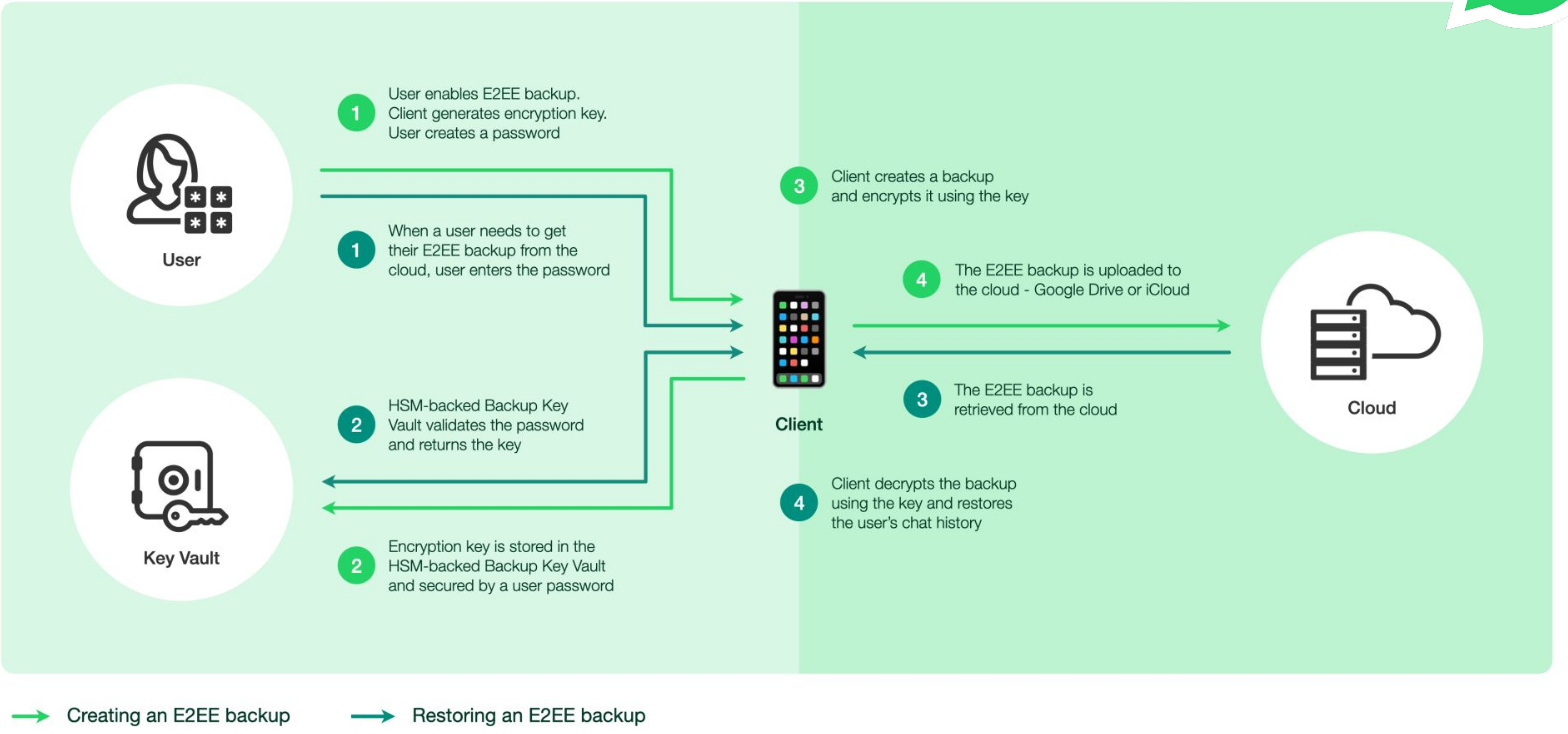
- Including from backup storage providers (Apple / Google)
- Including even from WhatsApp / Meta

Solutions:

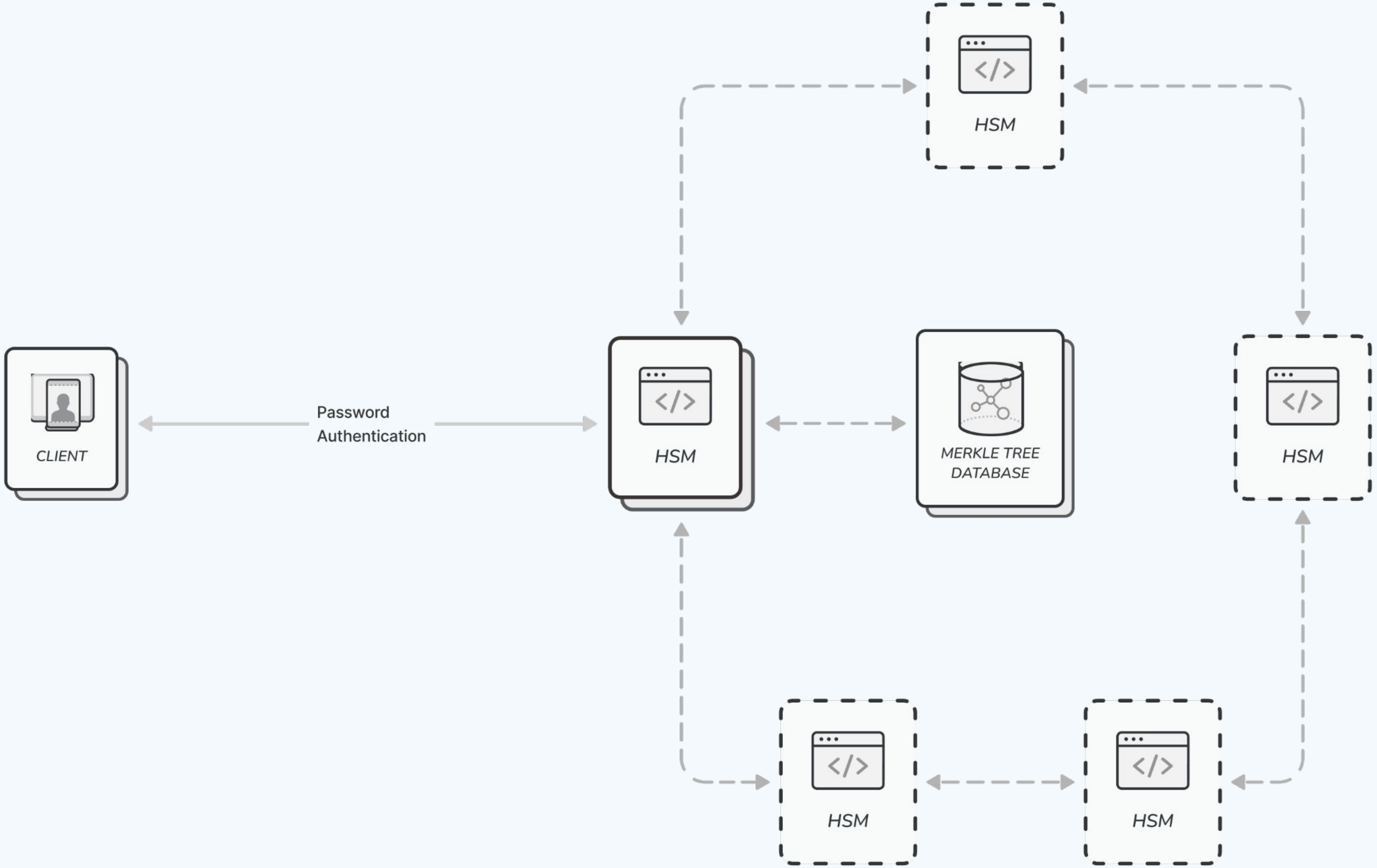
- Ask users to write down the bytes of their encryption key
- Ask users to remember a password + enforce attempt limit

This talk

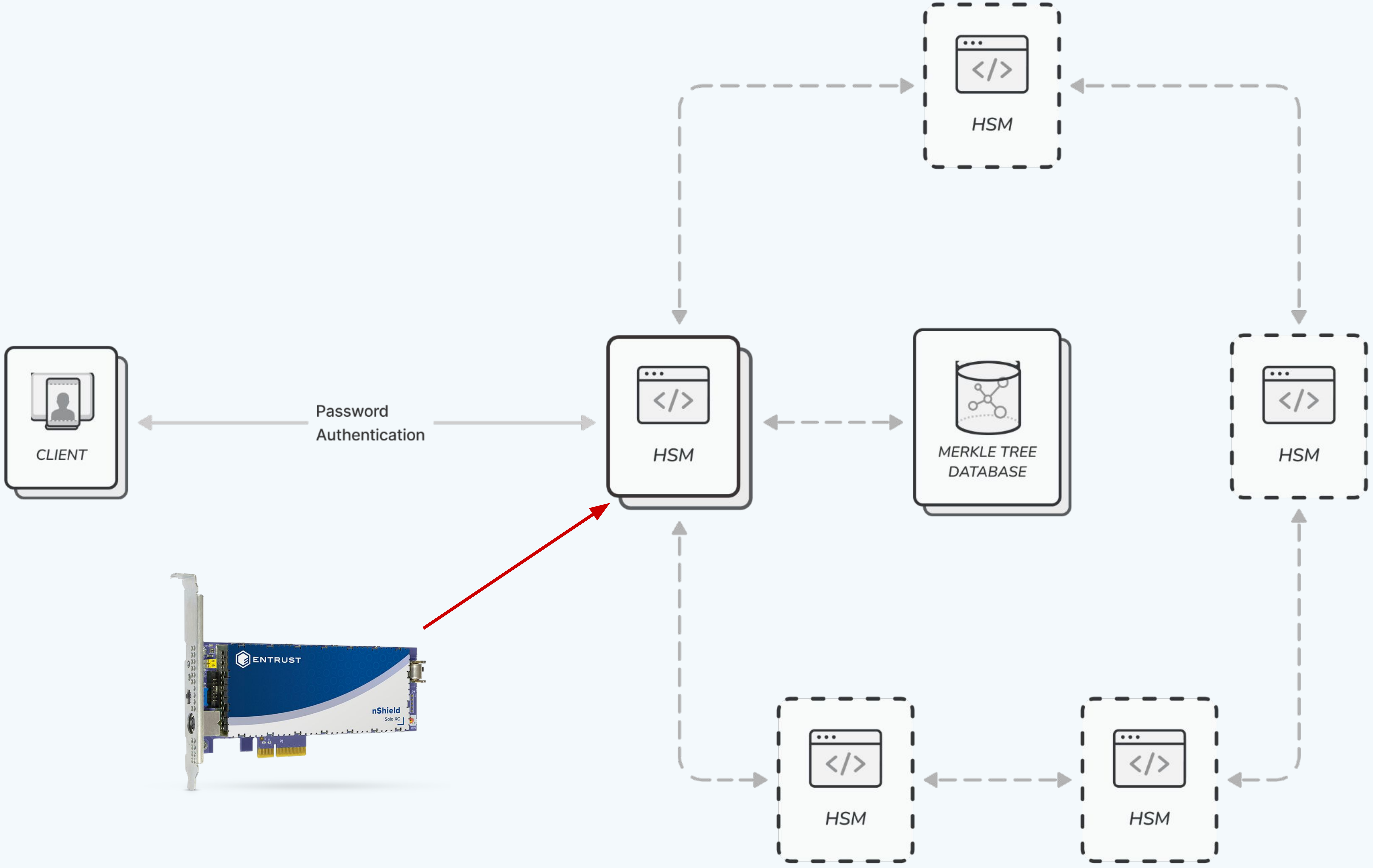
E2EE backup: User password



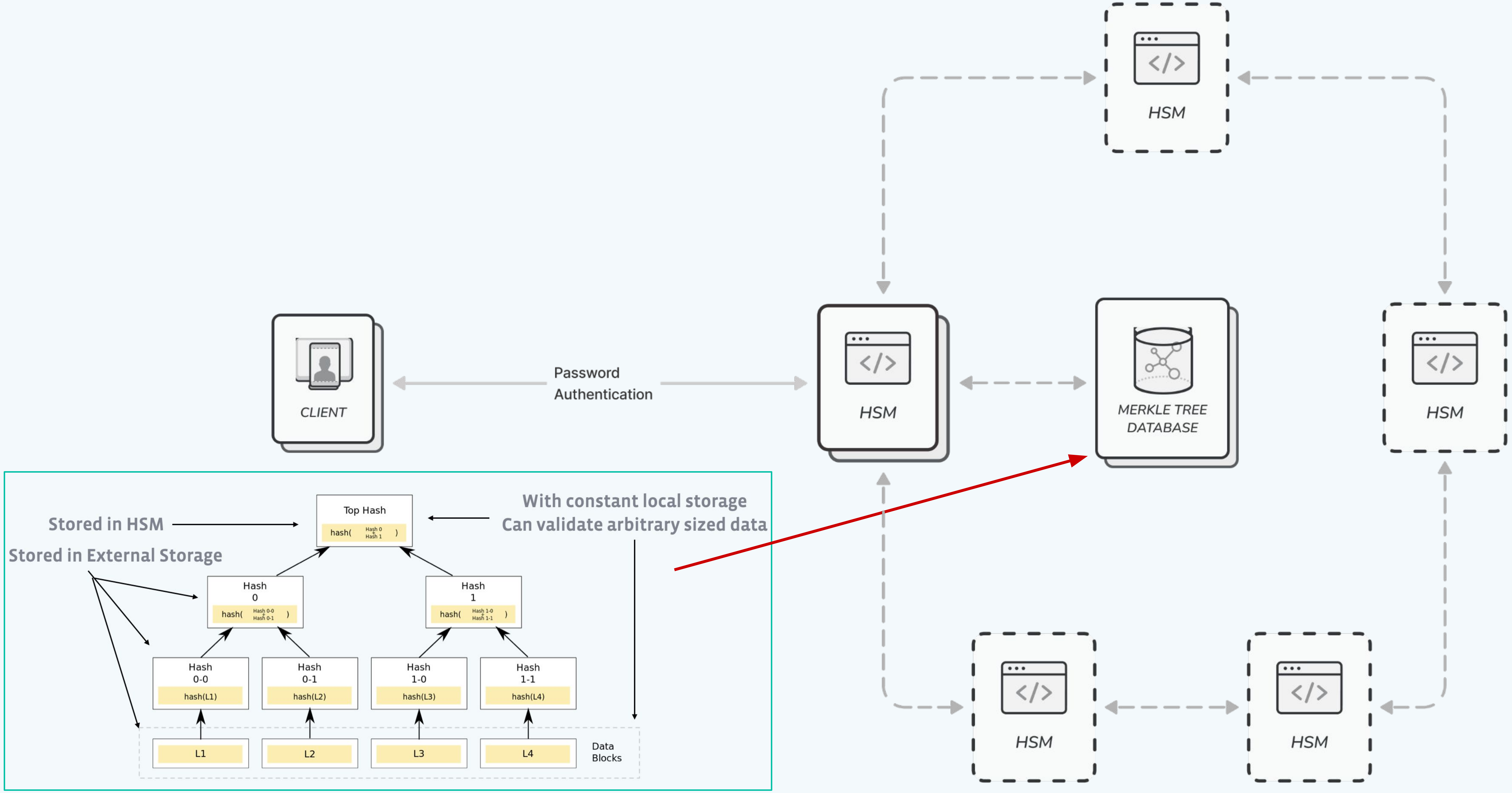
WhatsApp Key Vault



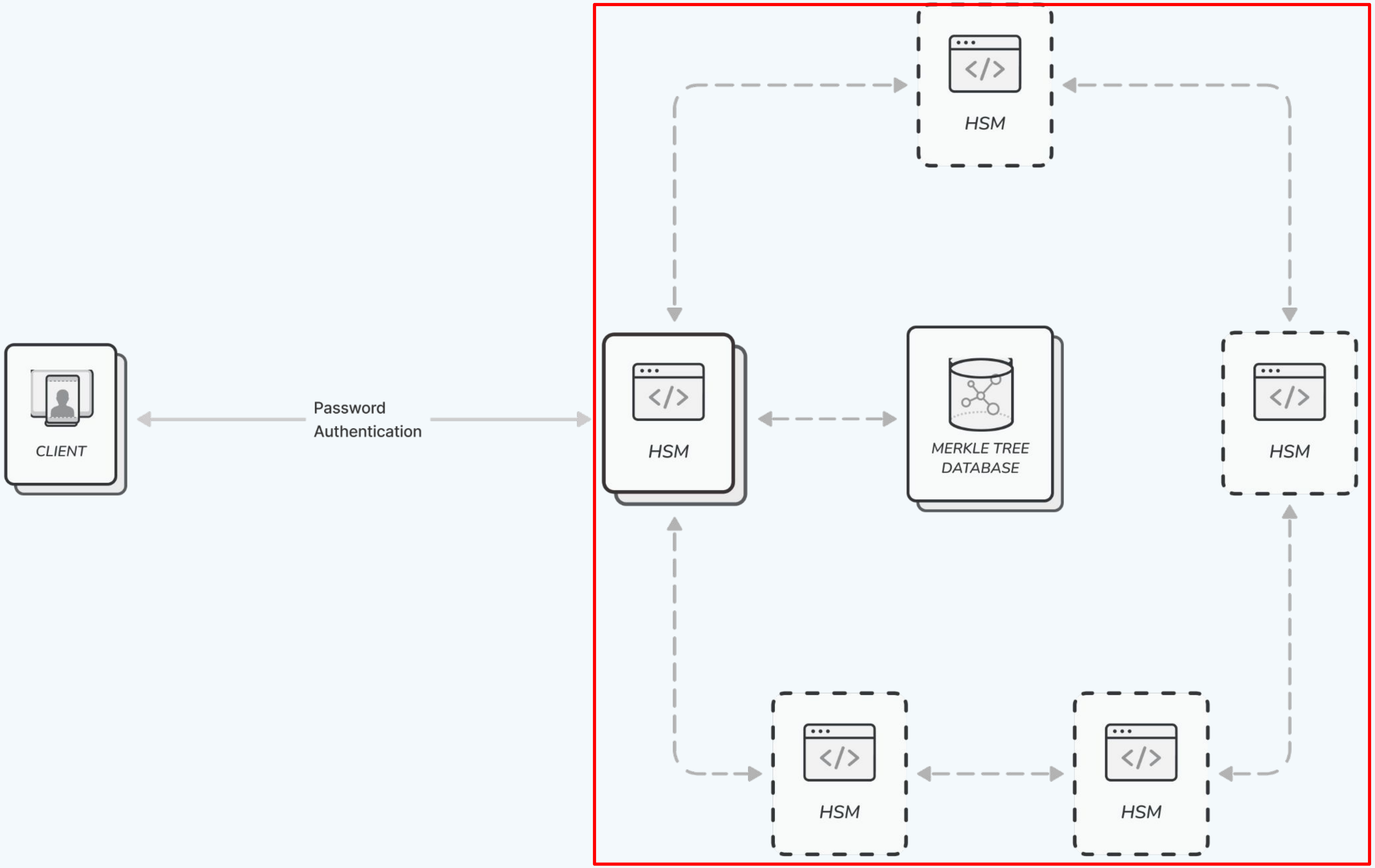
WhatsApp Key Vault



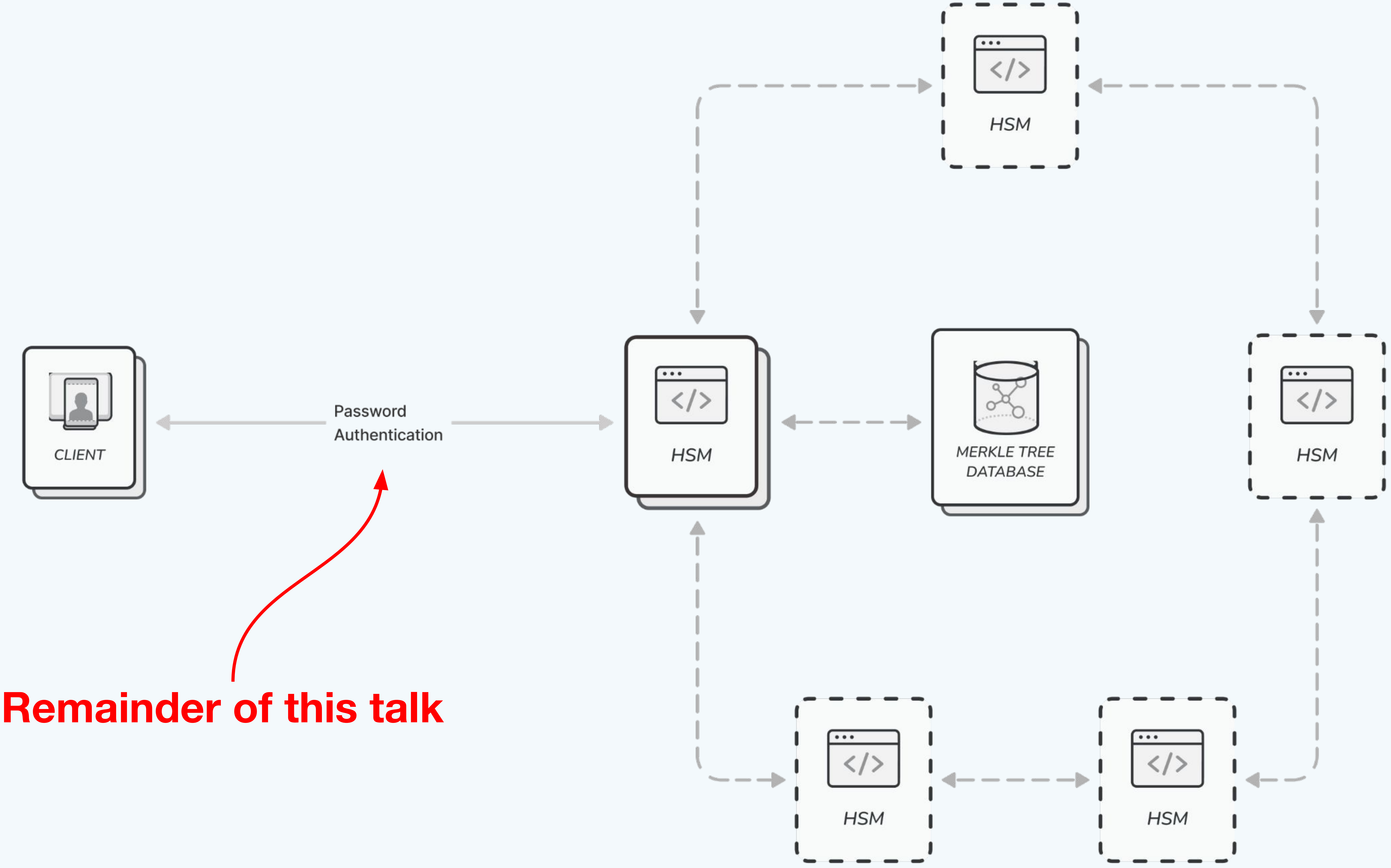
WhatsApp Key Vault



WhatsApp Key Vault



WhatsApp Key Vault



Client Authentication

How to securely authenticate client to HSMs?

Option 1: “Hash-then-Encrypt” or “Password-over-TLS”

- Client hashes their PIN, and then PK-encrypts to the HSM
- HSM decrypts and verifies hash

Option 2: Password-Authenticated Key Exchange

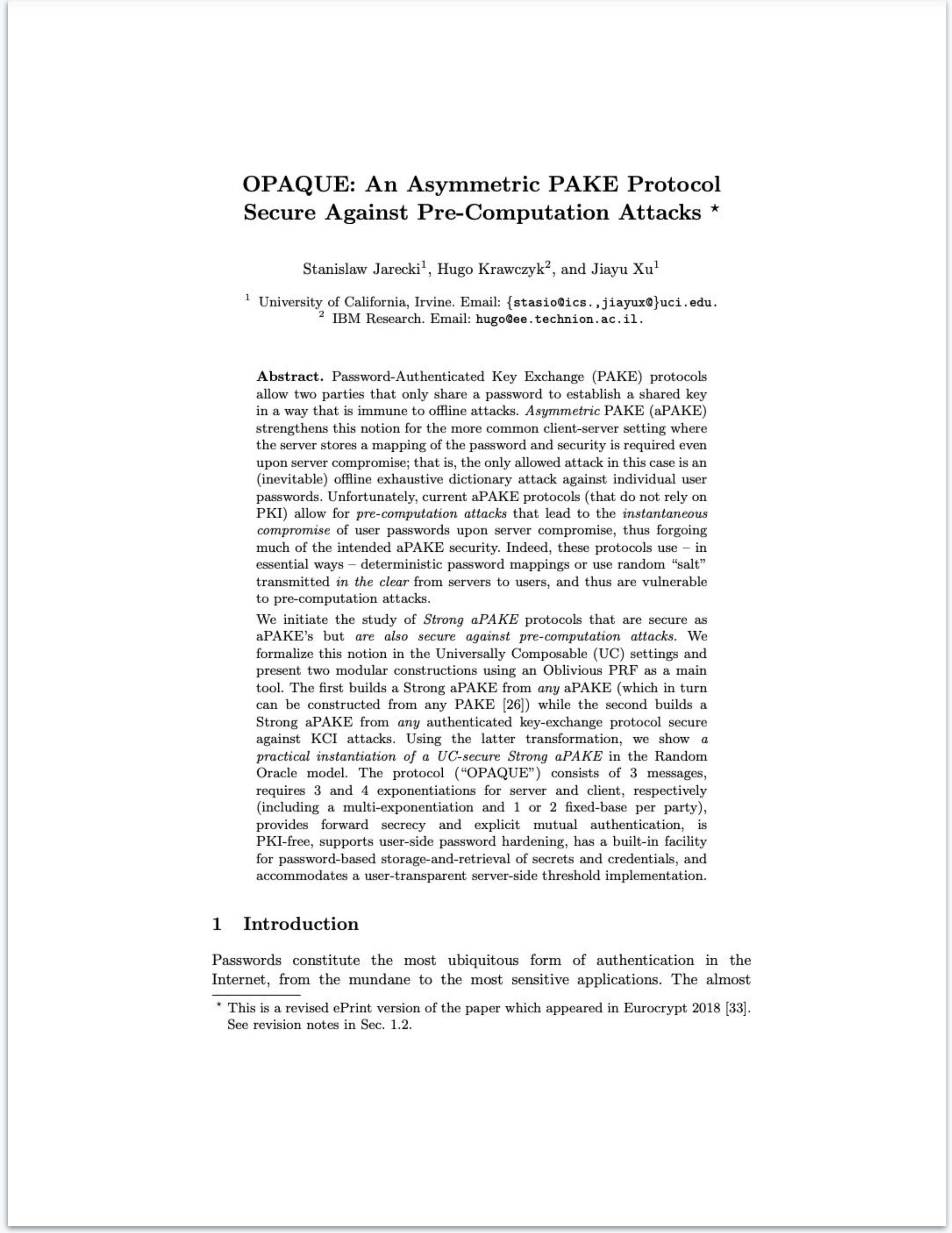
- Establish a secure channel between client and HSM based on PIN
- Transmit backup key through this channel

OPAQUE [Jarecki, Krawczyk, Xu ‘18]

- OPAQUE is a **strong**, *asymmetric* password-authenticated key exchange (**s***a*PAKE) protocol

- Theorem: Oblivious PRF + Authenticated KE -> saPAKE

- We use DH-OPRF: F(k,x) = H(x, H(x)^k)



OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks ⋆

Stanislaw Jarecki[1], Hugo Krawczyk[2], and Jiayu Xu[1]

[1] University of California, Irvine. Email: {stasio@ics.,jiayux@}uci.edu.
[2] IBM Research. Email: hugo@ee.technion.ac.il.

Abstract. Password-Authenticated Key Exchange (PAKE) protocols allow two parties that only share a password to establish a shared key in a way that is immune to offline attacks. *Asymmetric* PAKE (aPAKE) strengthens this notion for the more common client-server setting where the server stores a mapping of the password and security is required even upon server compromise; that is, the only allowed attack in this case is an (inevitable) offline exhaustive dictionary attack against individual user passwords. Unfortunately, current aPAKE protocols (that do not rely on PKI) allow for *pre-computation attacks* that lead to the *instantaneous compromise* of user passwords upon server compromise, thus forgoing much of the intended aPAKE security. Indeed, these protocols use – in essential ways – deterministic password mappings or use random “salt” transmitted *in the clear* from servers to users, and thus are vulnerable to pre-computation attacks.

We initiate the study of *Strong aPAKE* protocols that are secure as aPAKE’s but *are also secure against pre-computation attacks.* We formalize this notion in the Universally Composable (UC) settings and present two modular constructions using an Oblivious PRF as a main tool. The first builds a Strong aPAKE from *any* aPAKE (which in turn can be constructed from any PAKE [26]) while the second builds a Strong aPAKE from *any* authenticated key-exchange protocol secure against KCI attacks. Using the latter transformation, we show *a practical instantiation of a UC-secure Strong aPAKE* in the Random Oracle model. The protocol (“OPAQUE”) consists of 3 messages, requires 3 and 4 exponentiations for server and client, respectively (including a multi-exponentiation and 1 or 2 fixed-base per party), provides forward secrecy and explicit mutual authentication, is PKI-free, supports user-side password hardening, has a built-in facility for password-based storage-and-retrieval of secrets and credentials, and accommodates a user-transparent server-side threshold implementation.

1 Introduction

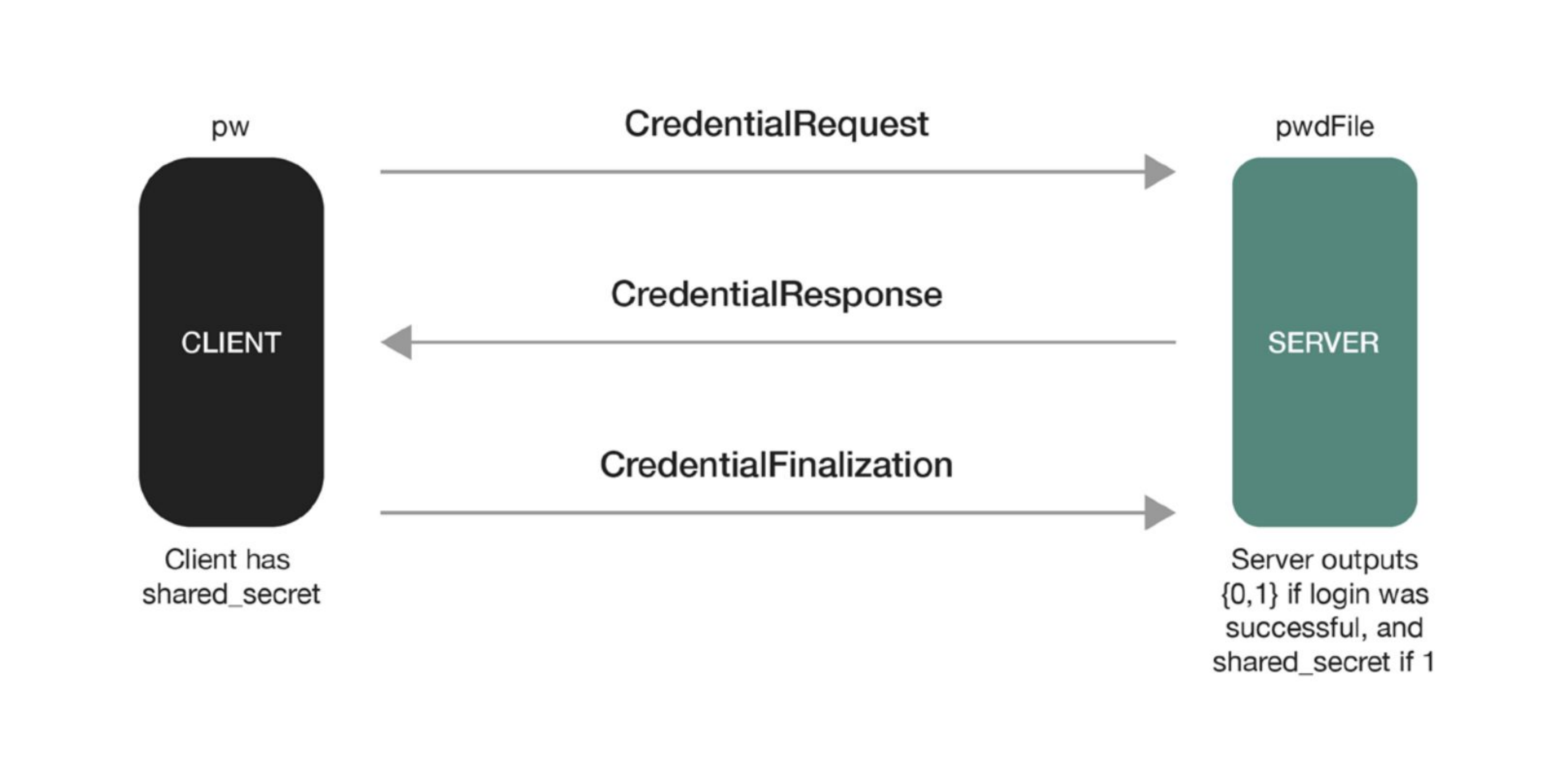
Passwords constitute the most ubiquitous form of authentication in the Internet, from the mundane to the most sensitive applications. The almost

⋆ This is a revised ePrint version of the paper which appeared in Eurocrypt 2018 [33]. See revision notes in Sec. 1.2.

OPAQUE in E2EE Backups

On backup registration:

On backup recovery:



E2EE Backups: Registration

K, pwd, HSM_PK

HSM_SK

Client

HSM Server

E2EE Backups: Registration

K, pwd, HSM_PK

HSM_SK

Client

HSM Server

1. Pick a random scalar r

2. Send α = H(pwd)^r

E2EE Backups: Registration

K, pwd, HSM_PK

HSM_SK

Client

HSM Server

1. Pick a random scalar r

2. Send α = H(pwd)^r

3. Pick a random OPRF key K’ and nonce

4. Send β = α^K’, nonce

E2EE Backups: Registration

K, pwd, HSM_PK

HSM_SK

Client

HSM Server

1. Pick a random scalar r

2. Send α = H(pwd)^r →

3. Pick a random OPRF key K’ and nonce

← 4. Send β = α^K’, nonce

5. Compute (export_key, client_SK) = PBKDF(pwd, β^(1/r))

6. Compute K* = AES-128(export_key, K) and client_PK = g^client_SK

E2EE Backups: Registration

K, pwd, HSM_PK

HSM_SK

Client

HSM Server

1. Pick a random scalar r

2. Send α = H(pwd)^r →

3. Pick a random OPRF key K’ and nonce

← 4. Send β = α^K’, nonce

5. Compute (export_key, client_SK) = PBKDF(pwd, β^(1/r))

6. Compute K* = AES-128(export_key, K) and client_PK = g^client_SK

7. Send E = RSA-OAEP(HSM_PK, K* || client_PK || transcript) →

8. Decrypt E and verify transcript, then store K*, K’, and client_PK for user

E2EE Backups: Registration

K, pwd, HSM_PK

HSM_SK

Client

HSM Server

1. Pick a random sca

3. Pick a random OPRF key K' and nonce

PRF F(k, x) = PBKDF(x, H(x)^k)

Client has backup key K and PIN.

For each client, server stores:

- K', a freshly generated PRF key
- K* = AES128(F(K', pwd), K)
- client_PK

5. Compute (export_

6. Compute K* = AES

7. Send E = RSA-OAEP(HSM_PK, K* || client_PK || transcript)

8. Decrypt E and verify transcript, then store K*, K', and client_PK for user

E2EE Backups: Recovery

pwd, HSM_PK

HSM_SK, K', K*, client_PK

Client

HSM Server

E2EE Backups: Recovery

pwd, HSM_PK

HSM_SK, K', K*, client_PK

Client

HSM Server

1. Pick a random scalar r and client_e_SK

$$\xrightarrow{\text{2. Send } \alpha = H(pwd)^{r}\text{, client\_e\_PK}}$$

E2EE Backups: Recovery

pwd, HSM_PK

HSM_SK, K’, K*, client_PK

Client

HSM Server

1. Pick a random scalar r and client_e_SK

2. Send α = H(pwd)^r, client_e_PK

3. Pick a server_e_SK, decrement attempt_counter

4. Send β = α^K’, server_e_PK, σ = Sign(HSM_SK, β)

E2EE Backups: Recovery

pwd, HSM_PK

HSM_SK, K’, K*, client_PK

Client

HSM Server

1. Pick a random scalar r and client_e_SK

2. Send $\alpha = H(pwd)^{\wedge}r$, client_e_PK →

3. Pick a server_e_SK, decrement attempt_counter

4. Send $\beta = \alpha^{\wedge}K'$, server_e_PK, $\sigma = \mathrm{Sign}(HSM\_SK, \beta)$ ←

5. Compute (export_key, client_SK) = PBKDF(pwd, $\beta^{\wedge}(1/r)$)

6. Derive shared_secret_key from KE protocol

7. < *Complete KE with server* > →

E2EE Backups: Recovery

pwd, HSM_PK

HSM_SK, K', K*, client_PK

<u>Client</u>

<u>HSM Server</u>

1. Pick a random scalar r and client_e_SK

2. Send α = H(pwd)^r, client_e_PK →

3. Pick a server_e_SK, decrement attempt_counter

4. Send $\beta = \alpha$^K', server_e_PK, σ = Sign(HSM_SK, β) ←

5. Compute (export_key, client_SK) = PBKDF(pwd, β^(1/r))

6. Derive shared_secret_key from KE protocol

7. < *Complete KE with server* > →

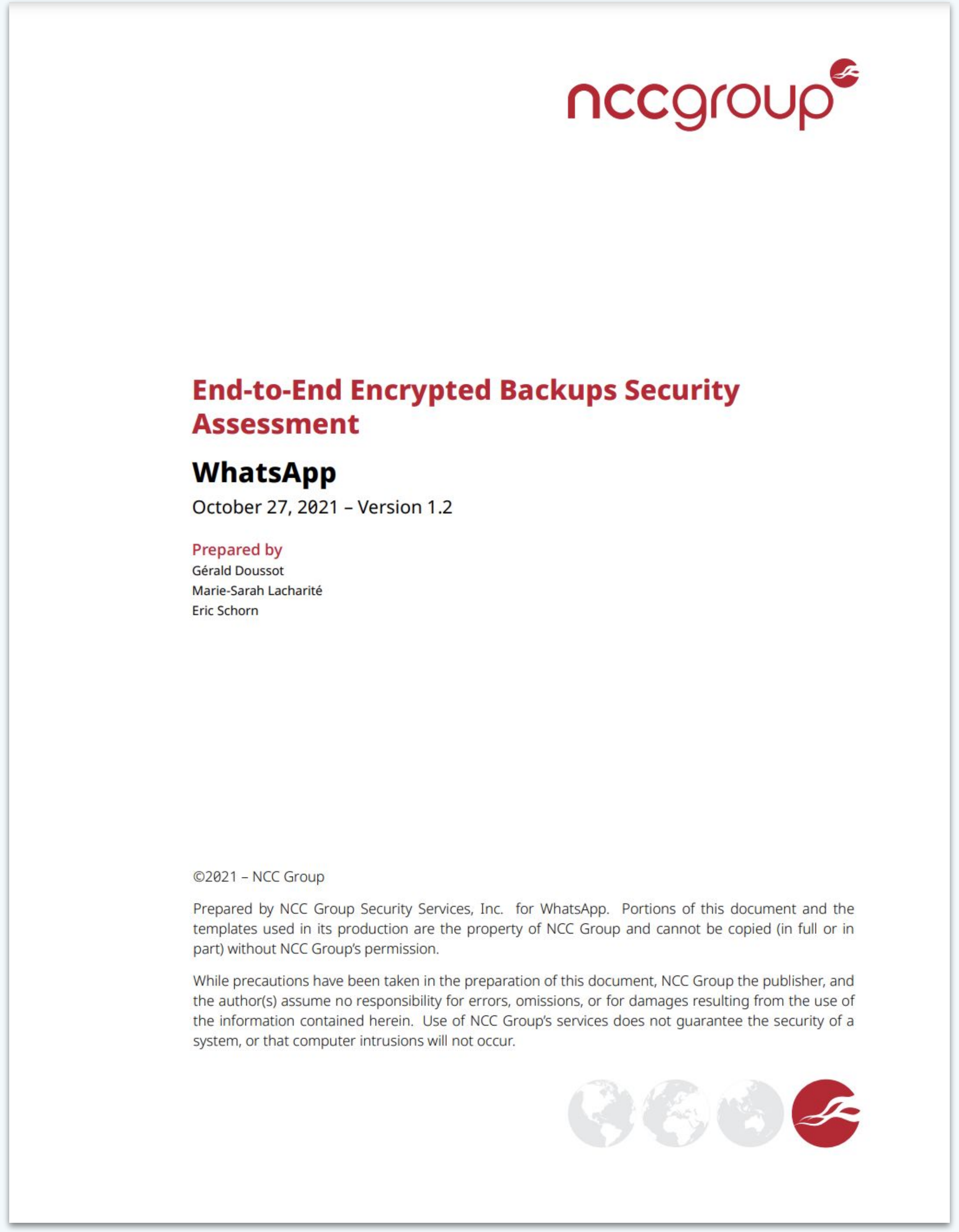
8. Verify KE completion, reset attempt_counter, and obtain shared_secret_key

9. Send C = AES(shared_secret_key, K*) ←

10. Decrypt C with shared_secret_key, then decrypt result with export_key to obtain K

More Resources

- Security audit from NCC Group in 2021
- We released a whitepaper on the E2EE backup design
- Open-source Rust OPAQUE library



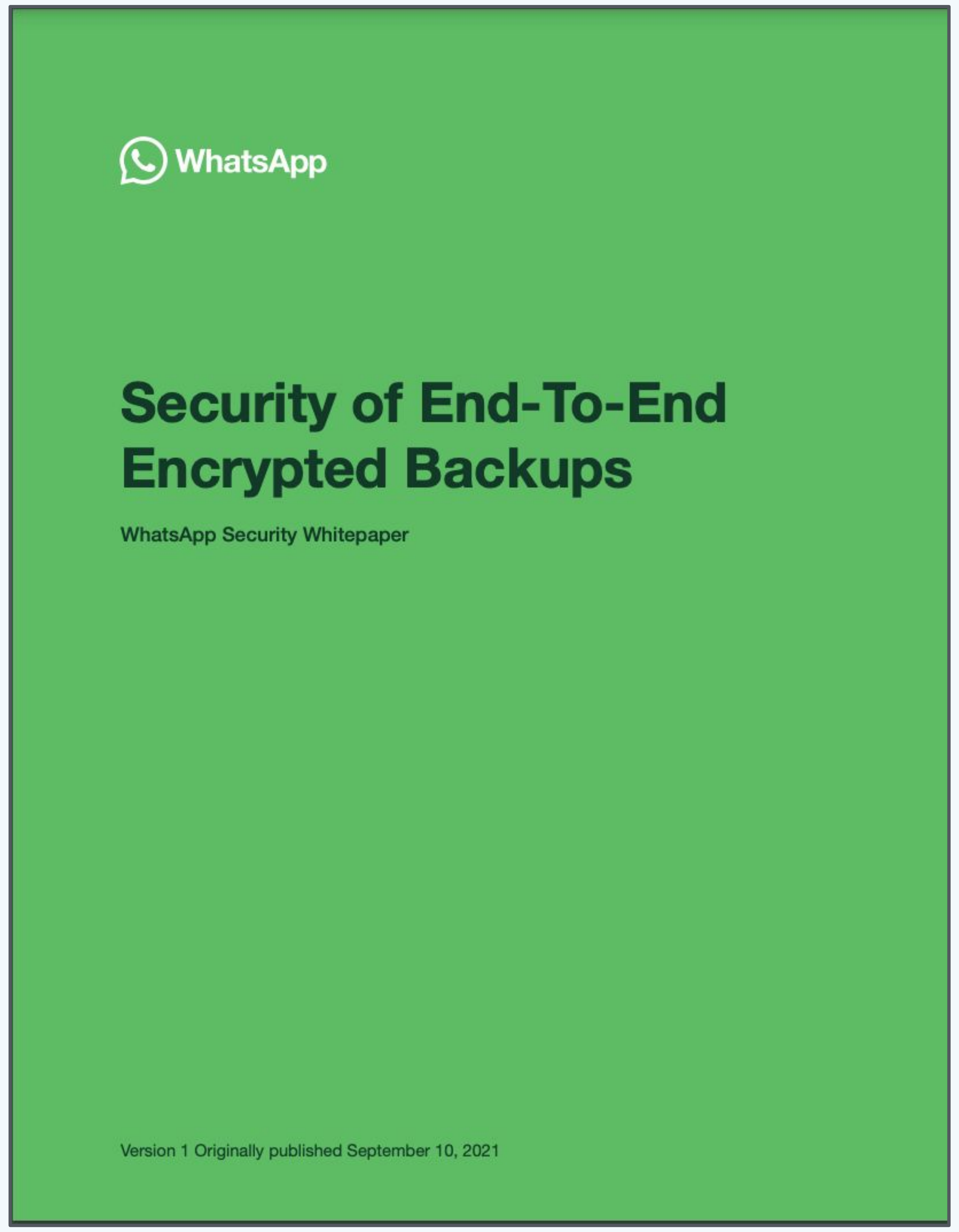
nccgroup

End-to-End Encrypted Backups Security Assessment

WhatsApp

October 27, 2021 – Version 1.2

Prepared by
Gérald Doussot
Marie-Sarah Lacharité
Eric Schorn



WhatsApp

Security of End-To-End Encrypted Backups

WhatsApp Security Whitepaper

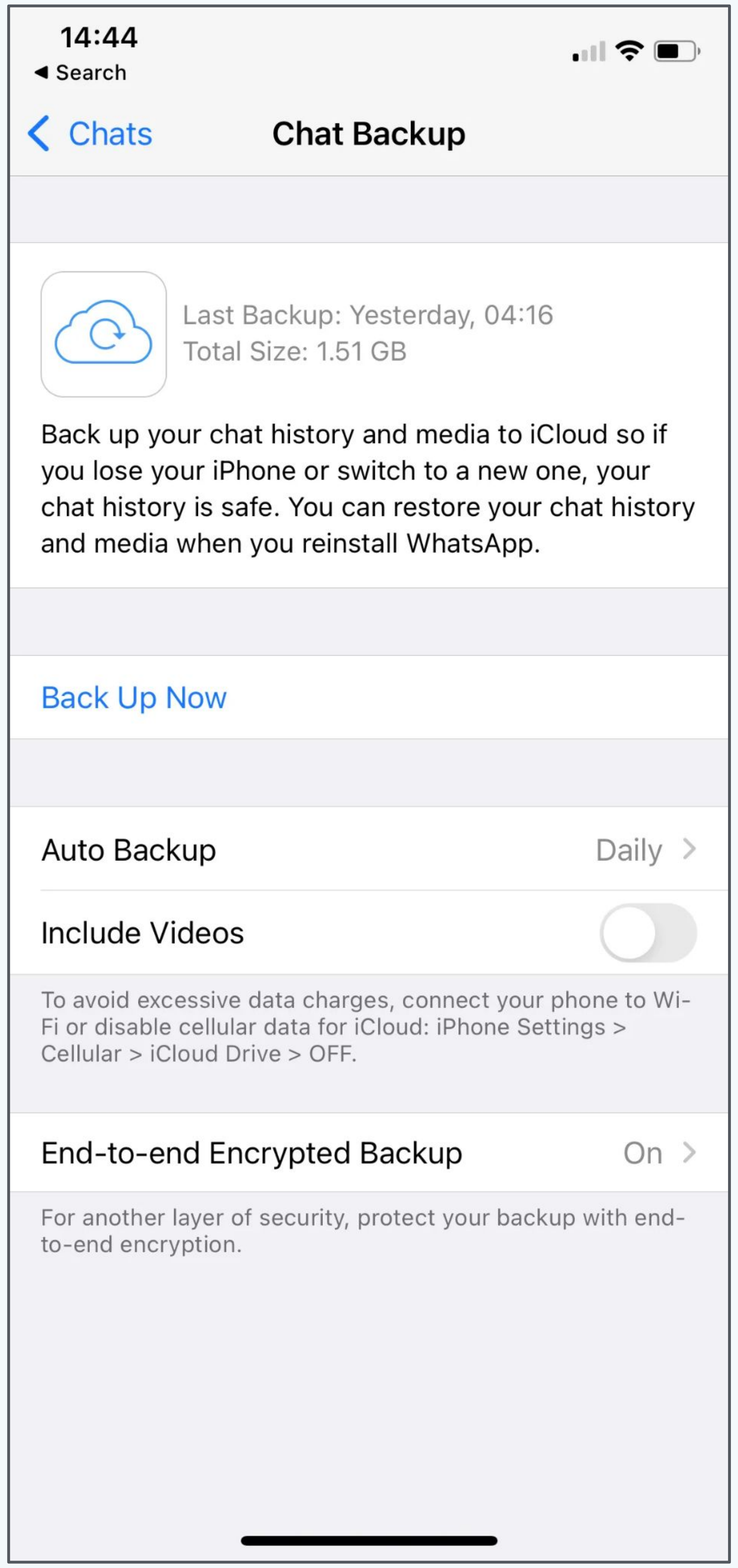
Version 1 Originally published September 10, 2021

Future Work

1. Alternatives to HSMs?

2. Threshold OPRFs / OPAQUE?

WhatsApp End-to-End Encrypted Backups



As of December 2022: