*

COSE Algorithms for Two-Party Signing

draft-lundberg-cose-two-party-signing-algs

Mike Jones, Emil Lundberg
IETF 122, Bangkok
March 19, 2025

Why Two-Party Signing?

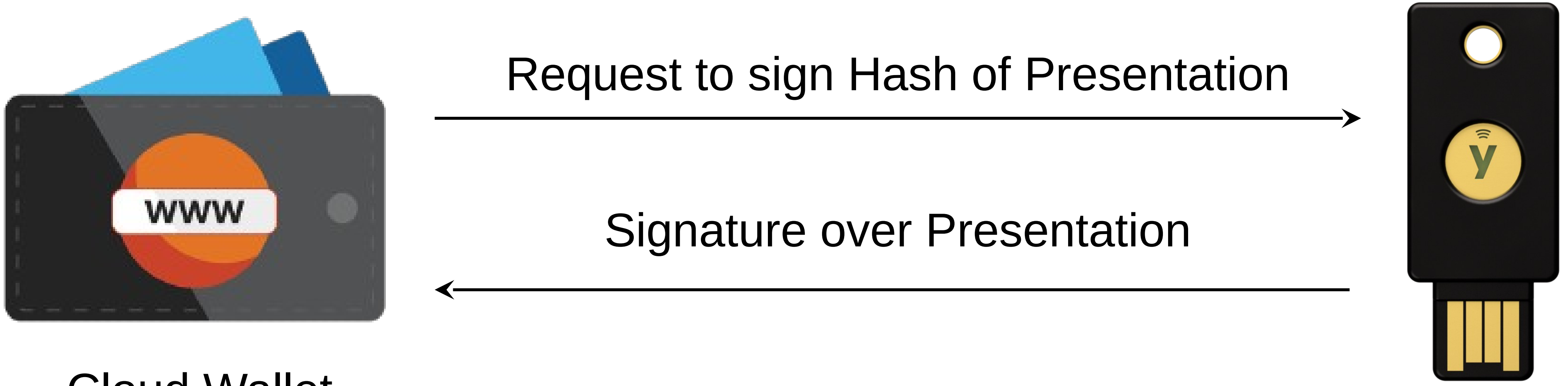
- Signing data too large to send to HSM holding private key
  - For example, cat video too large to send to FIDO authenticator to sign
  - Communication may be over restricted channels such as NFC, BLE
    - (Counterexamples: WYSIWYS w/ HSM display; blockchain h/w wallets)
- The good news
  - Many signing algorithms actually internally contain two steps
    1. Hash the data to be signed
    2. Use private key to sign the hashed data
  - Therefore, signing can be performed cooperatively by two parties
    1. Application does the hashing (the “*digester*”)
    2. HSM uses private key to sign the hashed data (the “*signer*”)
- Signature verified in one step using standard algorithms

The Name “Two-Party Signing”

- Not multi-party (M-of-N, etc.) signatures!
- Need better name?

Motivating Use Case

- Signing arbitrary data with WebAuthn/FIDO2 authenticators
  - See proposed [WebAuthn “sign” Extension]()
  - WebAuthn: `sign(authData || SHA256(fn(clientData)))`
  - Want: `sign(data)`

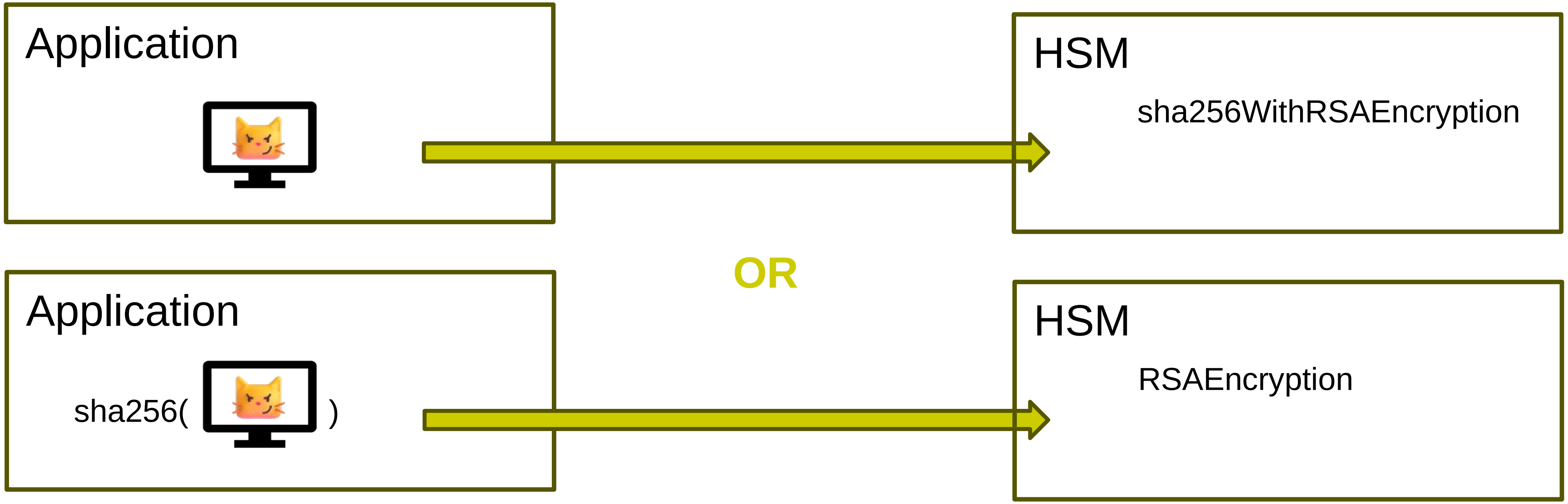


Cloud Wallet

Authenticator

Background: Pre-hash Modes

- With RSA or ECDSA, you are free to split the pre-hash step from the core signature step



- Both “modes” produce the same output, so this is just “implementation detail”

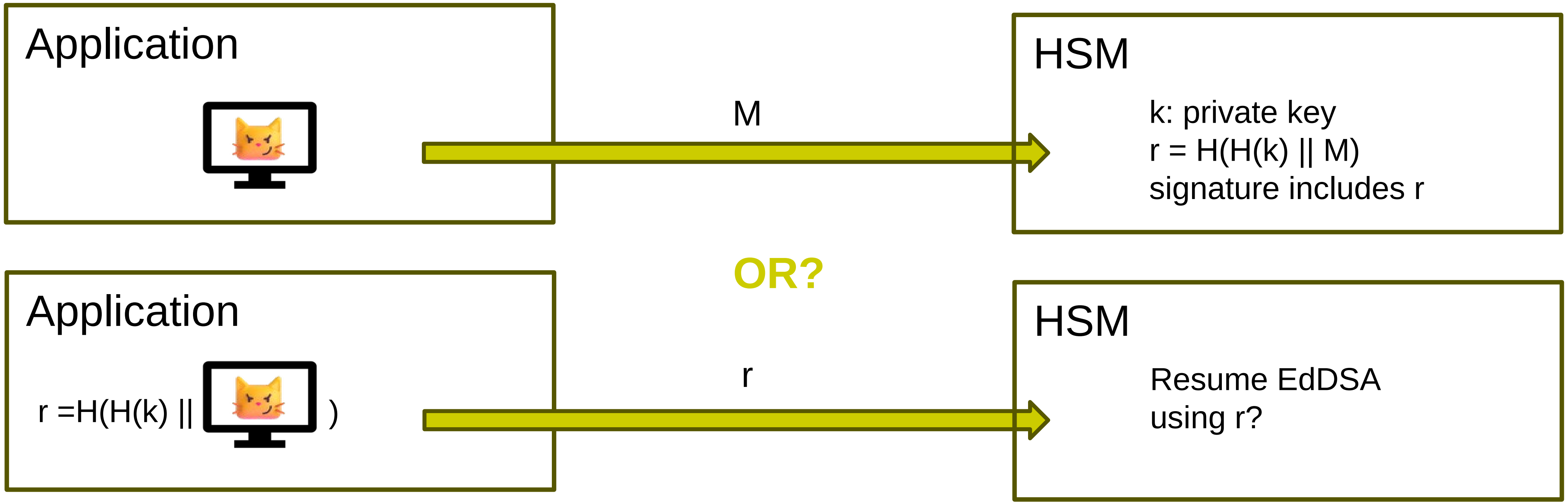
Slide by Mike Ounsworth – Used with permission

Why New Algorithm IDs?

- Cannot define “split point” generically
  - Obvious for ECDSA/RSA: “*digester* just does the hash”
  - Nontrivial for ECSDSA (Schnorr): hashes over *signer*-chosen nonce
  - Nontrivial for ML-DSA: hashes over ctx prefix and public key
    - Arguably, *ctx* prefix should be controlled by *signer* (private key holder)?
    - See also [LAMPS: ExternalMu-ML-DSA]()
  - Ambiguous for HashML-DSA: multiple hash steps
    - See also [LAMPS: Rationale for disallowing HashML-DSA]()
  - Impossible for PureEdDSA: hashes over private key
- => E.g., WebAuthn “sign” extension cannot generically specify “the data to be signed is pre-hashed by the application”
- => Use new alg IDs needed to signal how to process data to be signed

Caveat: Incompatible Algorithms

- For example, EdDSA hashes the message with the private key



- Application would need to know the private key k
  - Cannot split this algorithm

Slide based on original by Mike Ounsworth

ECDSA Two-Party Algorithms

Name	COSE Value	Base algorithm	Description
ESP256-2p	TBD	ESP256	ESP256 [I-D.jose-fully-spec-algs] divided for two-party signing as defined here
ESP384-2p	TBD	ESP384	ESP384 [I-D.jose-fully-spec-algs] divided for two-party signing as defined here
ESP512-2p	TBD	ESP512	ESP512 [I-D.jose-fully-spec-algs] divided for two-party signing as defined here

HashEdDSA Two-Party Algorithms*

Name	COSE Value	Base algorithm	Description
Ed25519ph-2p	TBD	Ed25519ph	Ed25519ph [I-D.jose-fully-spec-algs] divided for two-party signing as defined here (NOTE: Ed25519ph not yet registered)
Ed448ph-2p	TBD	Ed448ph	Ed448ph [I-D.jose-fully-spec-algs] divided for two-party signing as defined here (NOTE: Ed448ph not yet registered)

(* no current motivating use case)

HashML-DSA Algorithms*

Name	COSE Value	Base algorithm	Description
HashML-DSA-44-2p	TBD	HashML-DSA-44	HashML-DSA-44 divided for two-party signing as defined here (NOTE: HashML-DSA-44 not yet registered)
HashML-DSA-65-2p	TBD	HashML-DSA-65	HashML-DSA-65 divided for two-party signing as defined here (NOTE: HashML-DSA-65 not yet registered)
HashML-DSA-87-2p	TBD	HashML-DSA-87	HashML-DSA-87 divided for two-party signing as defined here (NOTE: HashML-DSA-87 not yet registered)

(* no current motivating use case)

Conveying Parameters from Digester to Signer

- Some algorithms have parameters used when signing
  - For example, ML-DSA has the parameter *ctx*
- For two-party signing, these parameters need to be conveyed from the *digester* to the *signer*
- We define “COSE key reference types” to enable this
  - Have fields for communicating these parameters between two parties
- Enables unified, algorithm-agnostic protocol between *digester* and *signer*

Example Key References

- Ref to ESP256 key

```
{
  1: TBD, ; kty: Ref-EC2
  2: h'35d189a78706147178f34b899ca5c35999d96ad51b5bff2fd7fa87fd60d2cba6’,
         ; kid: Opaque identifier of the EC2 key
  3: -9,  ; alg: ESP256 [Fully-Specified Algorithms for JOSE and COSE]
}
```

- Ref to ML-DSA-65 key supplying *ctx* parameter value

```
{
  1: TBD, ; kty: Ref-AKP
  2: h'92bc2bfa738f5bb07803fb9c0c58020791acd29fbe253baa7a03ac84f4b26d44’,
         ; kid: Opaque identifier of the AKP key
  3: TBD, ; alg: ML-DSA-65
 -1: 'Example application info’, ; ctx argument to ML-DSA.Sign
}
```

Next Steps

- Reviews requested!
  - Who is willing to volunteer?
- Create registrations for needed unregistered algorithms
  - `Ed25519ph`, `HashML-DSA-44`, etc.
- Are we missing anything?

Your Turn

- What else should we be considering?