*

Split Signing Algorithms for COSE

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

Mike Jones, Emil Lundberg

IETF 123, Madrid

July 21, 2025

What’s in a Name?

- Old name “COSE Algorithms for Two-Party Signing” could be misinterpreted
  - For instance, as multi-party (M-of-N, etc.) signatures
- New name “Split Signing Algorithms for COSE”
  - Abstract and Introduction updated accordingly

Changes Since IETF 122 in Bangkok

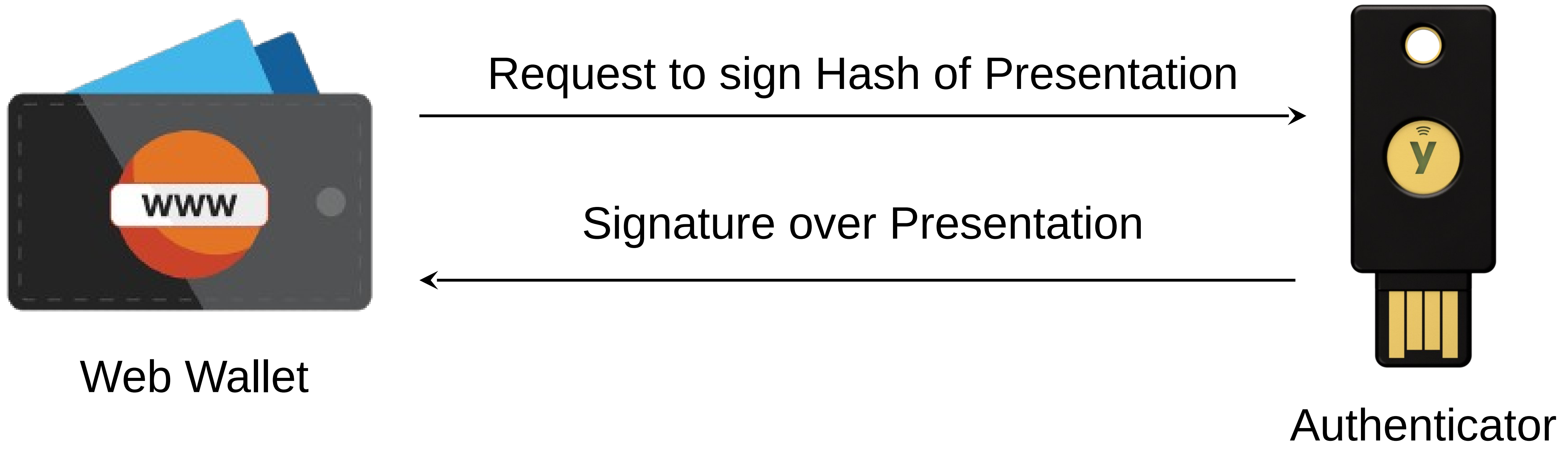
- New name, abstract, and introduction
- Dropped definitions for Hash ML-DSA
  - Split variants of ML-DSA are being discussed in other IETF groups
- Changed terminology from “Base Algorithm” to “Verification Algorithm”
- Replaced COSE_Key_Ref with COSE_Sign_Args
  - Used to convey additional arguments from the *digester* to the *signer*
  - “Reference to key” bad abstraction for some concrete use cases (Split-BBS)
  - Dropped definitions of reference types for COSE Key Types registry

Why Split 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
- 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
- Idea not new – used with smart cards, for instance

Motivating Use Case

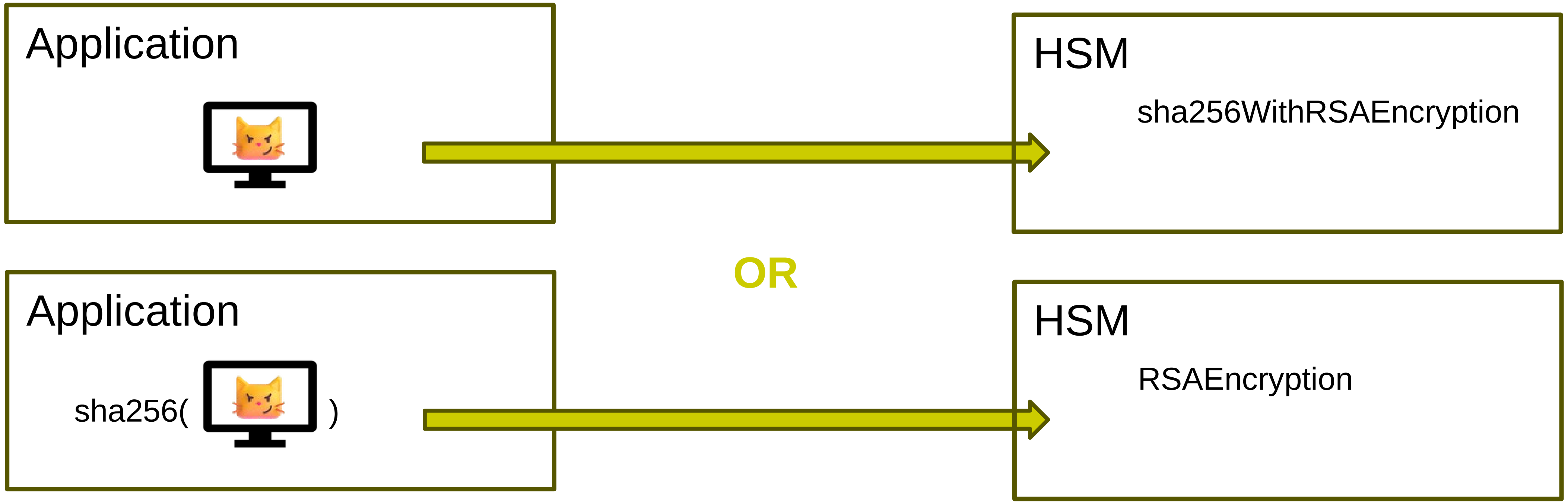
- Signing arbitrary data with WebAuthn/FIDO2 authenticators
  - See proposed [WebAuthn “sign” Extension]()
  - WebAuthn: `sign(authData || SHA256(fn(data)))`
  - Want: `sign(data)`



www
Request to sign Hash of Presentation
Signature over Presentation
y
Web 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”

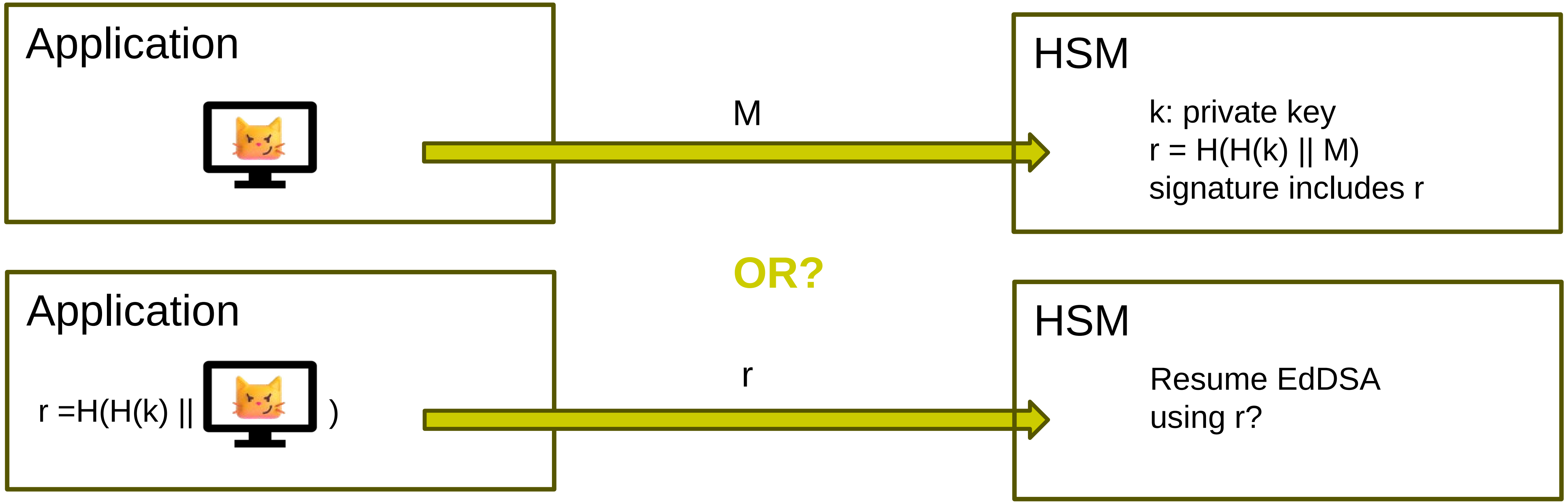
Slide by Mike Ounsworth – Used with permission

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 Split Signing Algorithms

Name	COSE Value	Verification Algorithm	Description
ESP256-split	TBD	ESP256	ESP256 [I-D.jose-fully-spec-algs] divided for split signing as defined here
ESP384-split	TBD	ESP384	ESP384 [I-D.jose-fully-spec-algs] divided for split signing as defined here
ESP512-split	TBD	ESP512	ESP512 [I-D.jose-fully-spec-algs] divided for split signing as defined here

HashEdDSA Split Signing Algorithms

Name	COSE Value	Verification Algorithm	Description
Ed25519ph-split	TBD	Ed25519ph	Ed25519ph [I-D.jose-fully-spec-algs] divided for split signing as defined here (NOTE: Ed25519ph not yet registered)
Ed448ph-split	TBD	Ed448ph	Ed448ph [I-D.jose-fully-spec-algs] divided for split signing as defined here (NOTE: Ed448ph not yet registered)

Conveying Parameters from Digester to Signer

- Some algorithms have parameters used when signing
  - For example: ML-DSA has the parameter *ctx*, [ARKG] has *kh* and *ctx*
- For split signing, these parameters need to be conveyed from the *digester* to the *signer*
- We define COSE_Sign_Args structure to enable this
  - Have fields for communicating these parameters between two parties
- Enables unified, algorithm-agnostic protocol between *digester* and *signer*

[ARKG]: draft-bradleylundberg-cfrg-arkg

Example additional args for ESP256-split using key derived w/ ARKG-P256

```
{
  3: -65539,   ; alg: ESP256-split with ARKG-P256 (placeholder value)

               ; ARKG-P256 key handle
               ; (HMAC-SHA-256-128 followed by
                  SEC1 uncompressed ECDH public key)
  -1: h'27987995f184a44cfa548d104b0a461d
        0487fc739dbcdabc293ac5469221da91b220e04c681074ec4692a76ffacb9043de
          c2847ea9060fd42da267f66852e63589f0c00dc88f290d660c65a65a50c86361',

               ; ctx argument to ARKG-Derive-Private-Key
  -2: 'ARKG-P256.test vectors',
}
```

[ARKG]: draft-bradleylundberg-cfrg-arkg

Example additional args for “[Split BBS]”

```
{
  3: -65602,   ; alg: SplitBBS-BS256 (placeholder value)

  -10: {       ; t2prime argument to Split-BBS signing procedure
    1: 2,        ; kty: EC2
    -1: -65601,  ; crv: BLS12_381 (placeholder value)
                ; x coordinate of t2prime
    -2: h'19c902dfc093fe8165c98543dae09a3d4a9006dfb5ba1e6e
          46b7495f9384e4c26d1af74302ff95bc922d4b1649ed9630',
                ; y coordinate of t2prime
    -3: h'0ef6b57c2fdb550b33287728daed69637201eb53294cc82c
          304c3e3937fd1c7346e6da79d242c25ffa728130316130a5',
  },
}
```

- t2prime is a transaction binding value, not a key property => remodel COSE_Key_Ref as COSE_Sign_Args

[Split BBS]: https://yubicolabs.github.io/cose-two-party-signing-algs-rfc/split-bbs/draft-lundberg-cose-two-party-signing-algs.html#name-split-bbs

Next Steps

- Reviews requested!
  - Who is willing to volunteer?
- Create registrations for needed unregistered algorithms
  - `Ed25519ph`, etc.
- What else should we be considering?