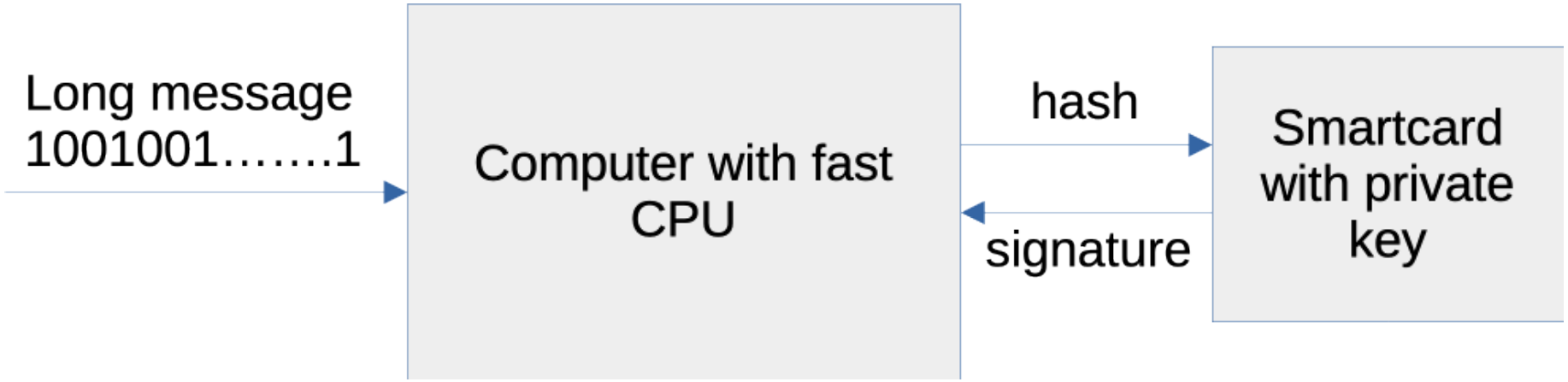
Prehashing Panel Discussion

Panelists:

- Scott Fluhrer, Cisco
- Markku-Juhani O. Saarinen, Tampere University and PQShield
- Joseph Harvey, Verisign

Moderator: John Kelsey, NIST and KU Leuven

What is prehashing?



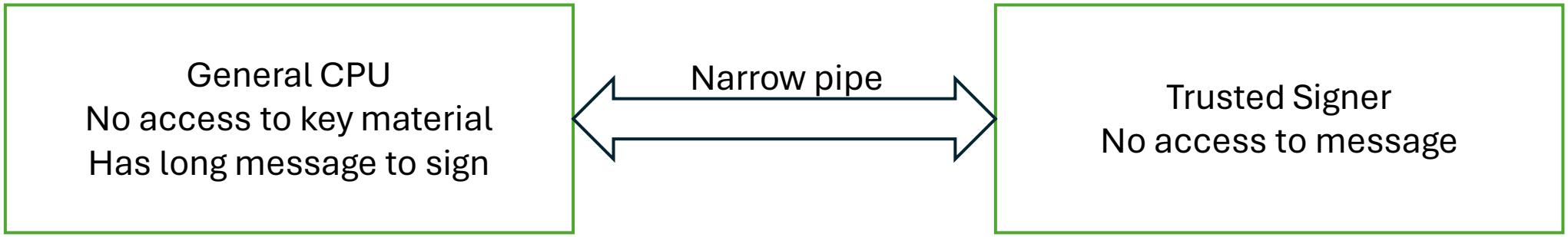
Normal signature:

- S = Sign(SK,M)
- Signing alg. processes the message with a hash
- May include randomizers, public key, etc.

Pre-hashing:

- H = hash(M), S = Sign(SK, H)
- Hashing takes place outside signing alg.
- Signing algorithm will do more hashing, randomize, etc.

Use case #1 for prehashing



Current practice:
- General CPU hashes the long message; sends the hash through the narrow pipe
- Trusted signer pads and RSA/ECDSA signs the message; sends the signature through the narrow pipe

Issue with ML-DSA – it insists that the hash used is prefixed with data from the public key; the General CPU doesn't have access to that.

Proposed solution: have the General CPU do a general hash; have the trusted signer do a full ML-DSA signature (including the hash) of the hash

Use case #2 for prehashing

We want to sign the same message with multiple algorithms (e.g. RSA, ML-DSA)

Issue – ML-DSA insists that the hash used is prefixed with data from the public key; RSA does not

Hence, we cannot use the same hash for both

If the message is long, the repeated message hashing is expensive.

Proposed solution: hash the message once, and then have each algorithm sign the hash.

Use case #3 for prehashing

We want to sign the long message with SLH-DSA

Issue – SLH-DSA requires two passes over the message:

- The first to determine an unpredictable ‘optrand’ value
- The second to hash the message (along with the ‘optrand’ value and public data)

If the message is long, the repeated message hashing is expensive.

Proposed solution: hash the message once (with a hash function we have confidence in its collision resistance), and then have SLH-DSA sign that hash (with its two passes)

Use case #4 for prehashing

We want to use a hash that’s not supported by crypto module

Examples:

- New, very efficient hash function + old crypto module
- Parallel hash or tree hash

What can go wrong?

If we just allowed Sign(SK,M) or Sign(SK, H), we'd get ambiguity

Sign(SK,X) might be

- Signature on the message X
- Signature on the message Y where X = hash(Y)
- How do we tell?

- Introduces a kind of dumb forgery attack we want to avoid.

Domain-separation to the rescue

Something like this—this isn't a full specification

M = message, H = externally computed hash, ctx = context

- Pre-hash:
  - H = hash(M) using SHAKE256 with 512-bit output
  - S = sign(0 || OID(SHAKE256 with 512-bit output) || ctx || H)
- Normal:
  - S = sign(1 || ctx || M)

Panel Questions

- Should FIPS 204 and 205 specify an optional pre-hashing step? Alternatively, should NIST provide guidance in a Special Publication?
- If not, should NIST encourage development of a general-purpose specification and/or guidance for pre-hashing in other standards development organizations?
- Or, would it be preferable to have special-purpose specifications and/or guidance developed by the protocols and use cases that employ a pre-hashing option?

NIST is currently planning to have one general pre-hashing scheme

- *Apply to all PQ sigs*

More panel questions

- Should randomized hashing be included as an option in the guidance or specification?
- What about other inputs, such as the signer's public key?

Current plan: Don't incorporate public key, randomizers, etc. into external hash.

Still more questions

- What are some examples of the protocols and use cases that might employ a pre-hashing option? What is their rationale?
- What other kinds of usage guidance for pre-hashing messages would be helpful to have?
- How will pre-hashing play with existing APIs?