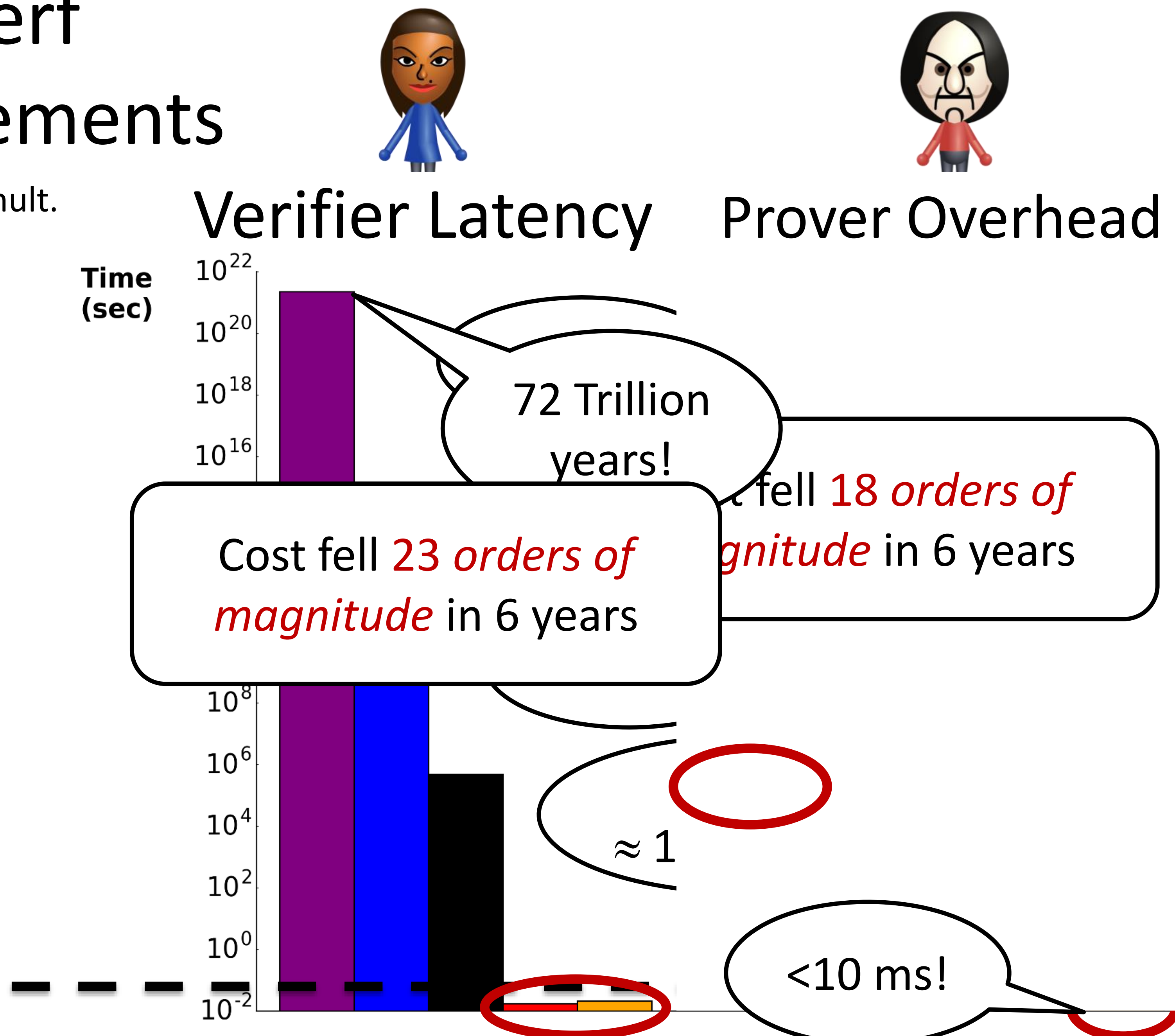
Making Verifiable Computation Useful

Bryan Parno

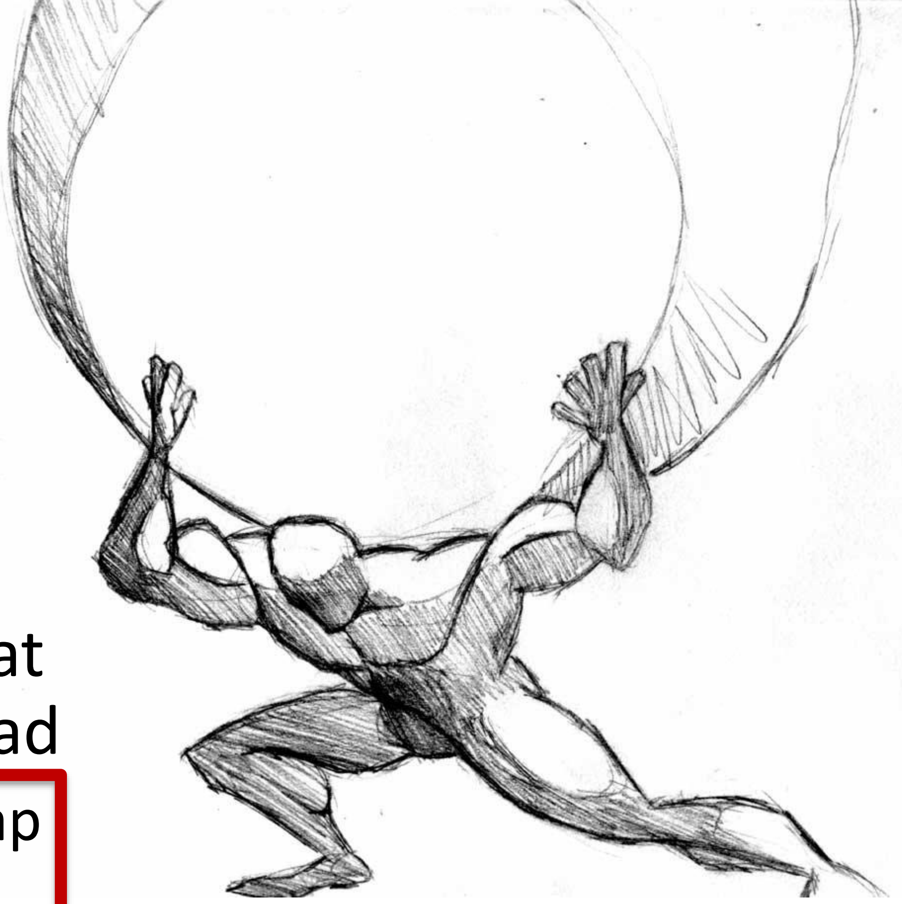
Carnegie Mellon University

Rapid Perf Improvements

100x100 matrix mult.



Coping with Prover Overhead



1. Leverage zero knowledge
   – Example: Bitcoin++
     [Danezis et al. '13] [Ben-Sasson et al. '14] [Kosba et al. '15] [Miller et al. '15]
2. Find (rare?) applications that tolerate substantial overhead
   – Original computation is cheap or infrequent
     - Example: Fair exchange of digital goods [Maxwell '16]

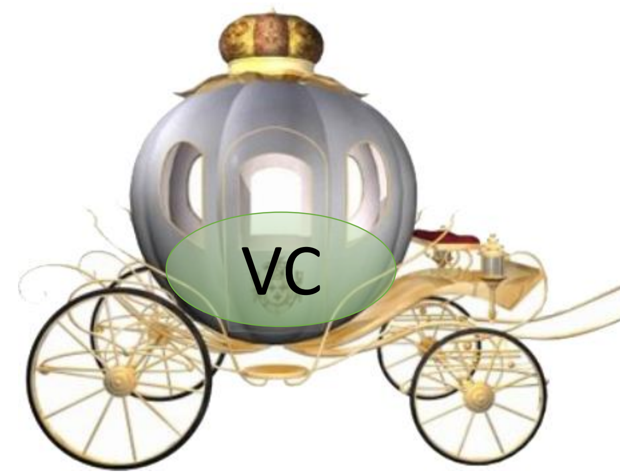
   – Integrity benefits outweigh costs
     - Example: Verifiable ASICs [Wahby et al. '15]
3. Innovations in proof generation

Cinderella: Turning Shabby X.509 Certificates into Elegant Anonymous Credentials

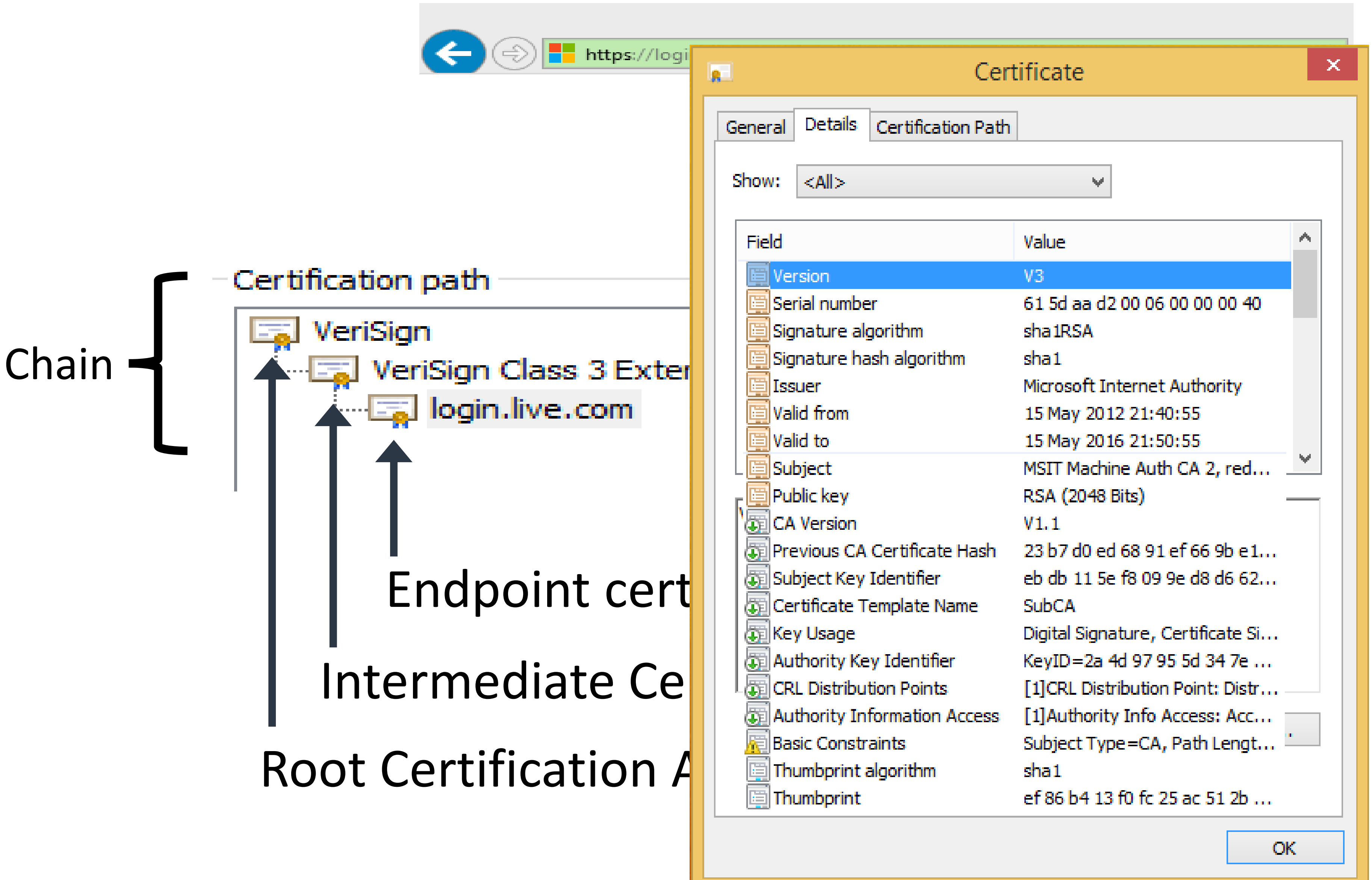
with the Magic of Verifiable Computation

[IEEE S&P '16]

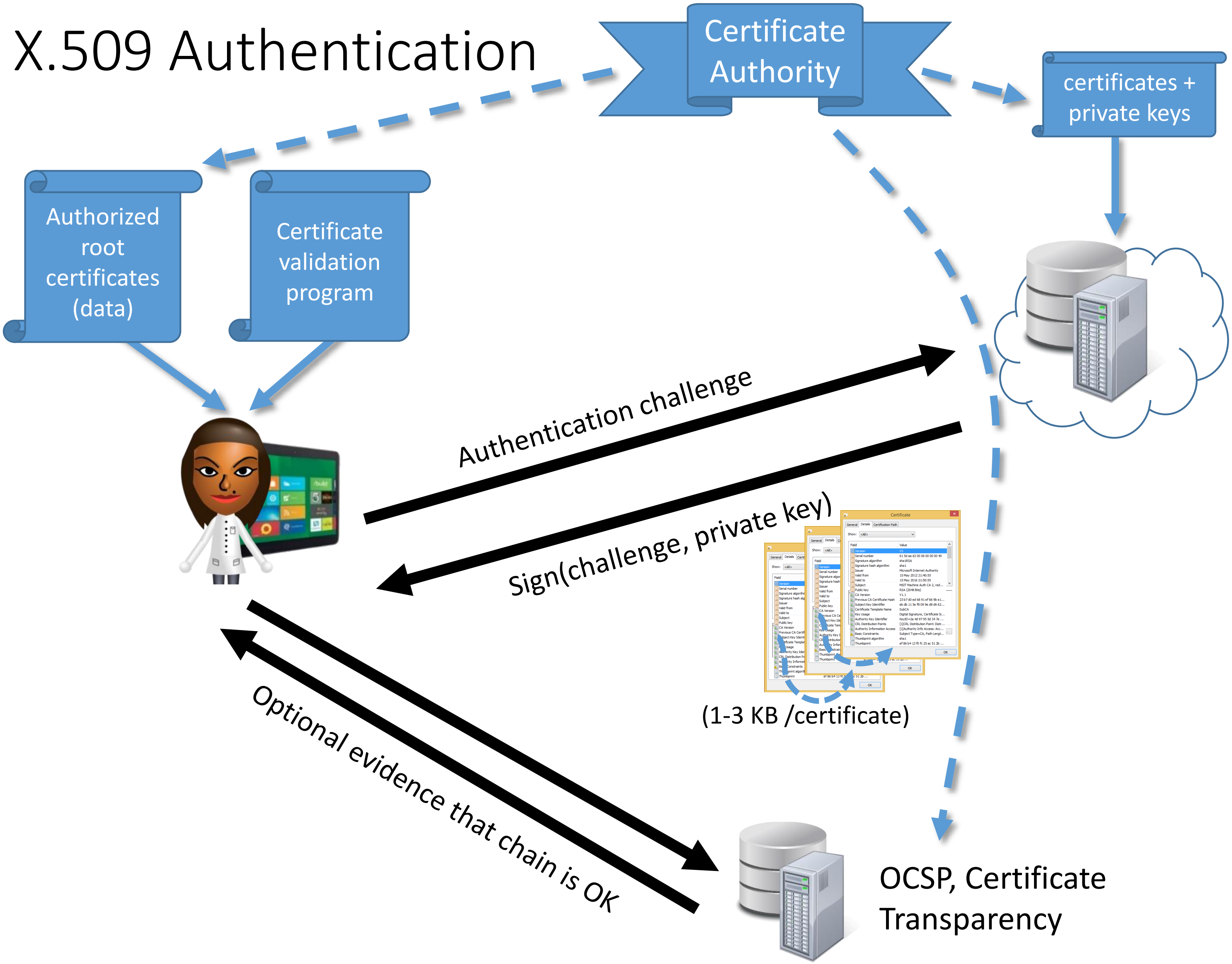
Antoine Delignat-Lavaud
Cédric Fournet
Markulf Kohlweiss
Bryan Parno



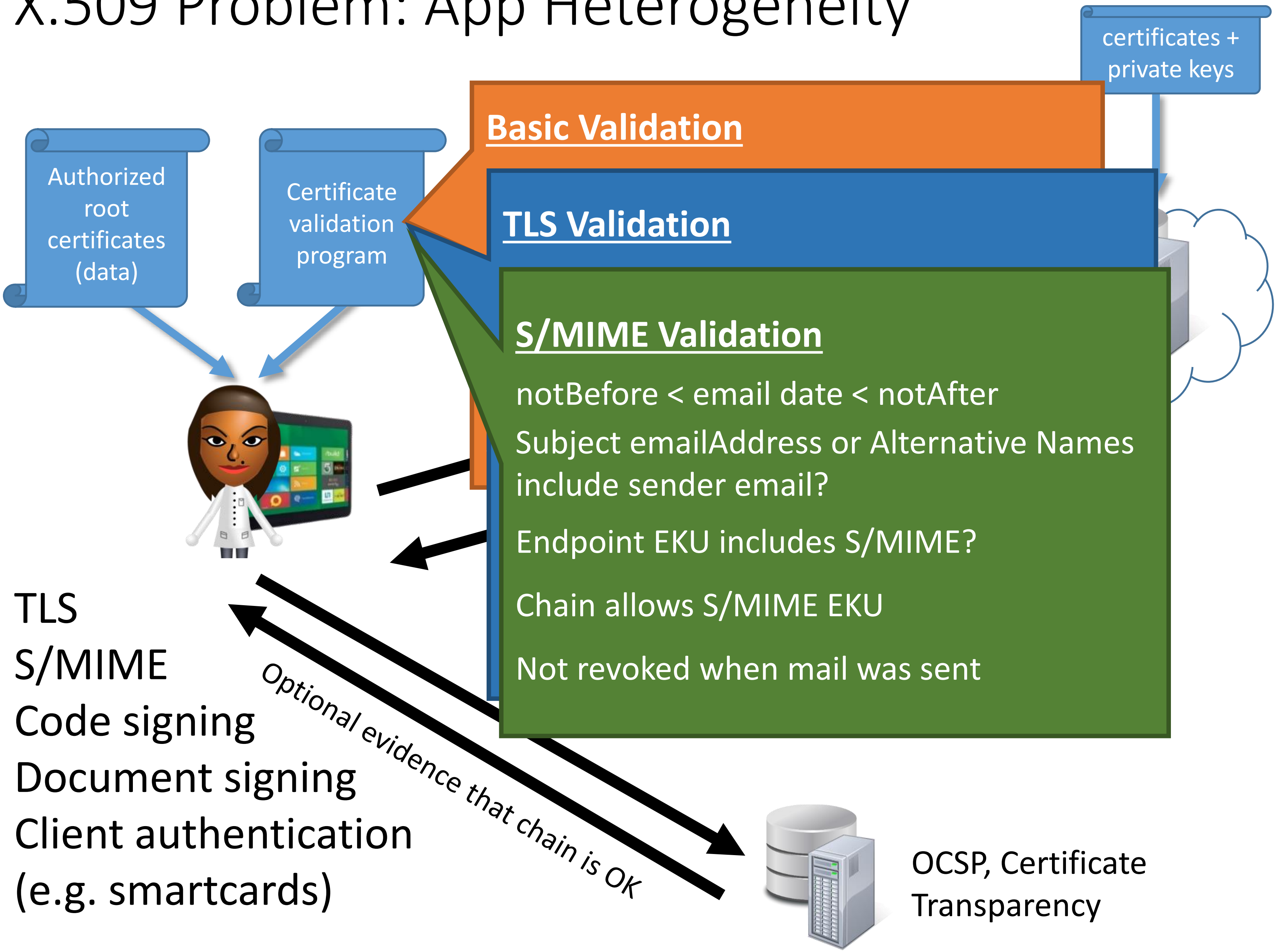
The X.509 Public Key Infrastructure (1988)



X.509 Authentication

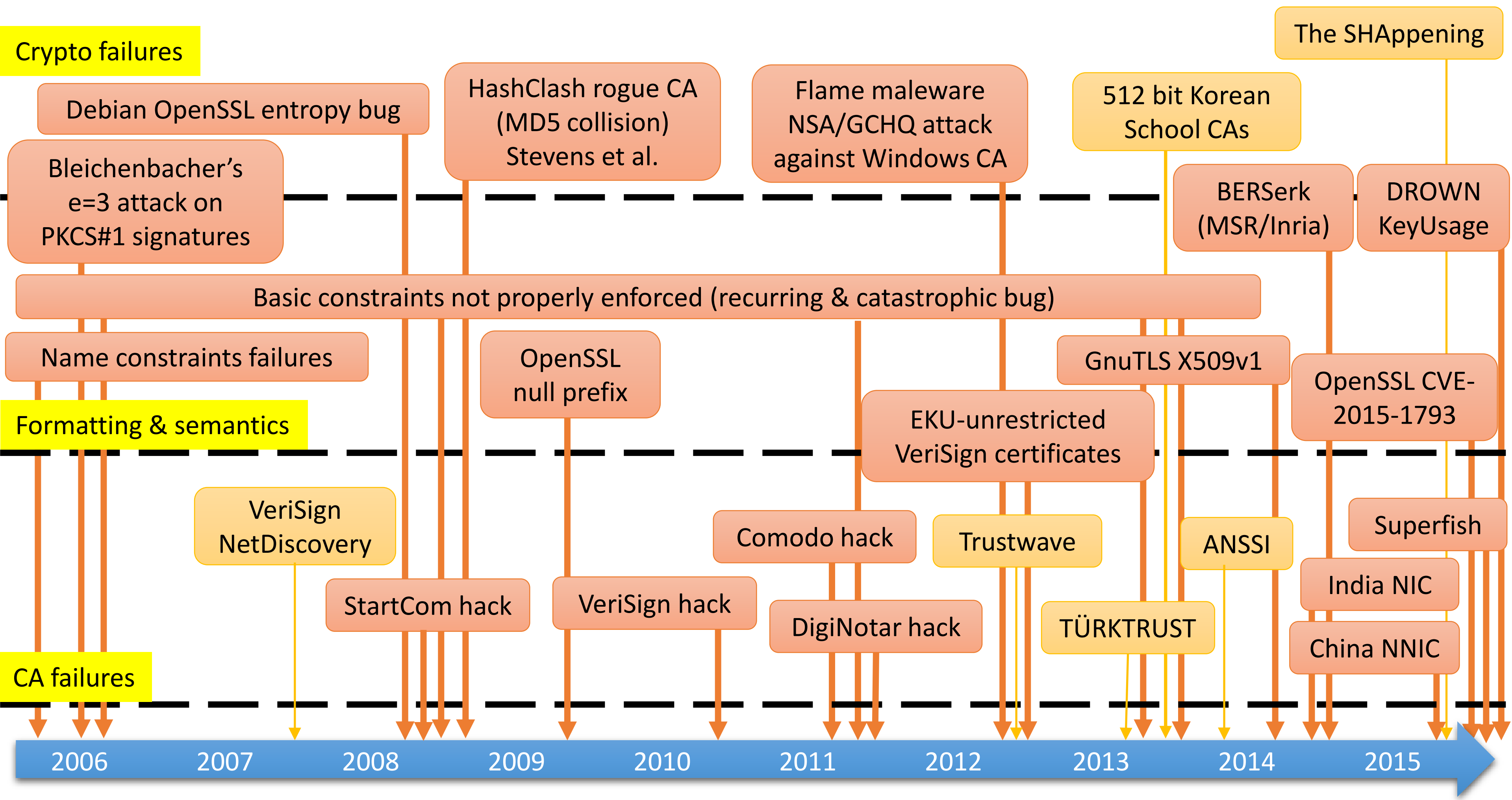


X.509 Problem: App Heterogeneity

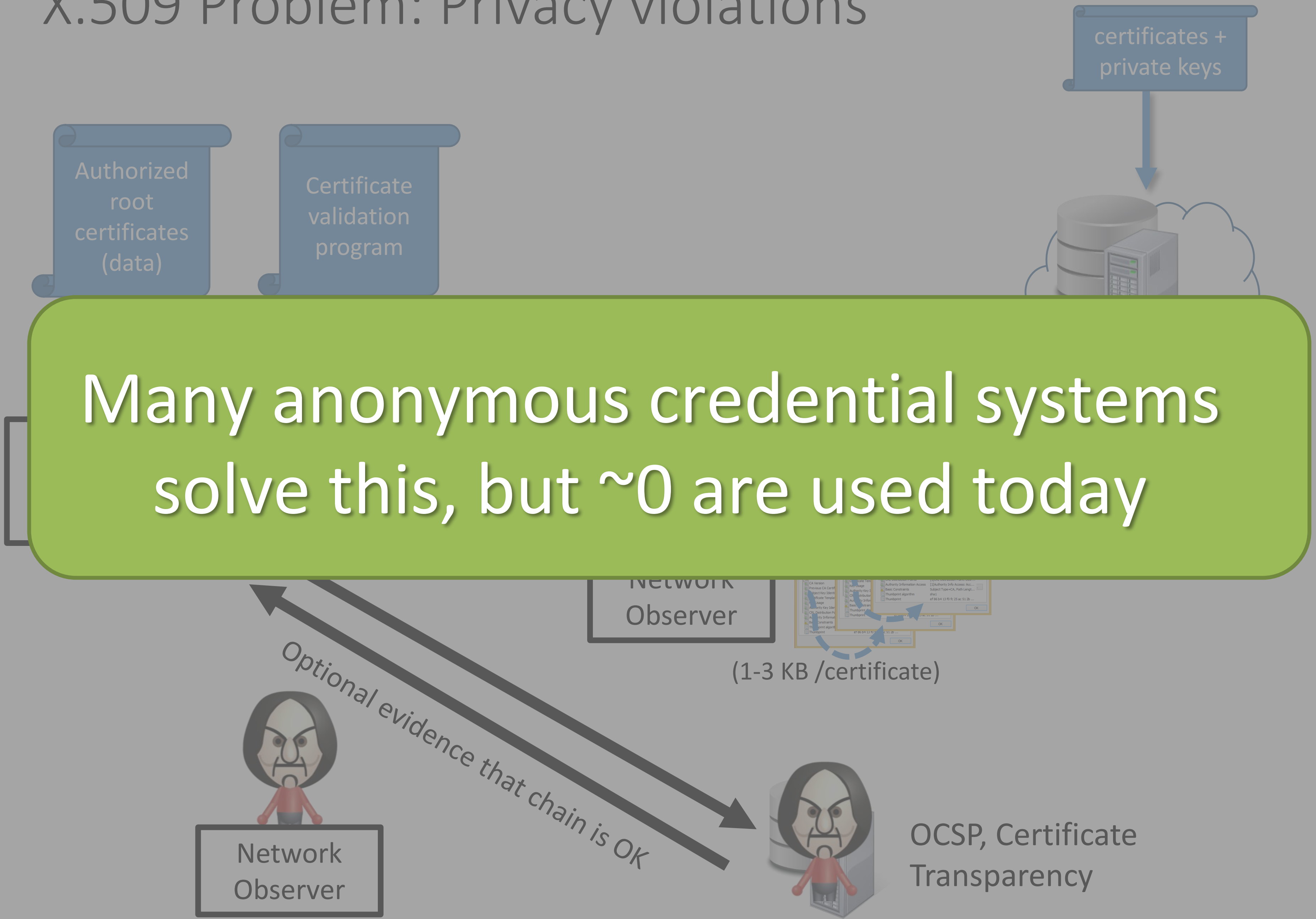


- TLS
- S/MIME
- Code signing
- Document signing
- Client authentication (e.g. smartcards)
- …

Recent PKI Failures

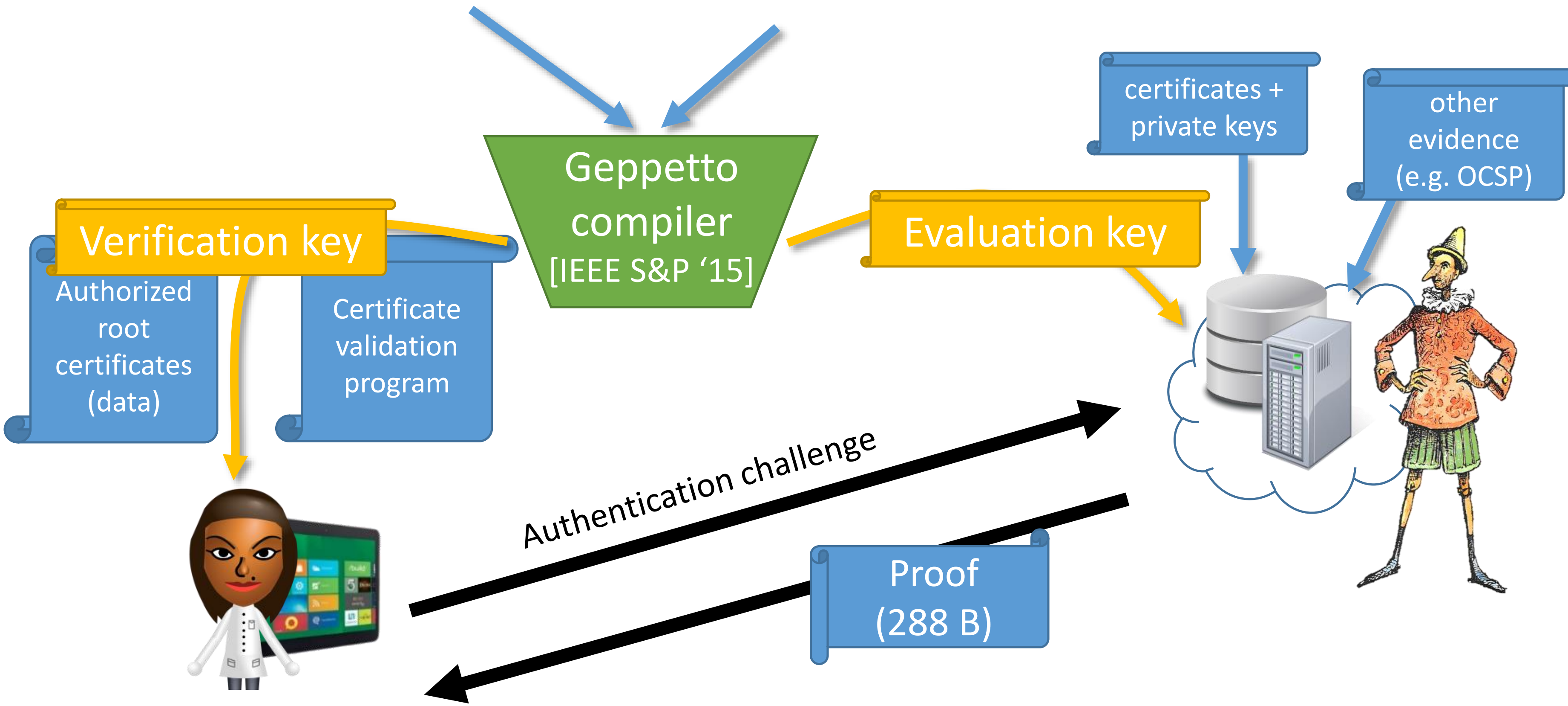


X.509 Problem: Privacy violations

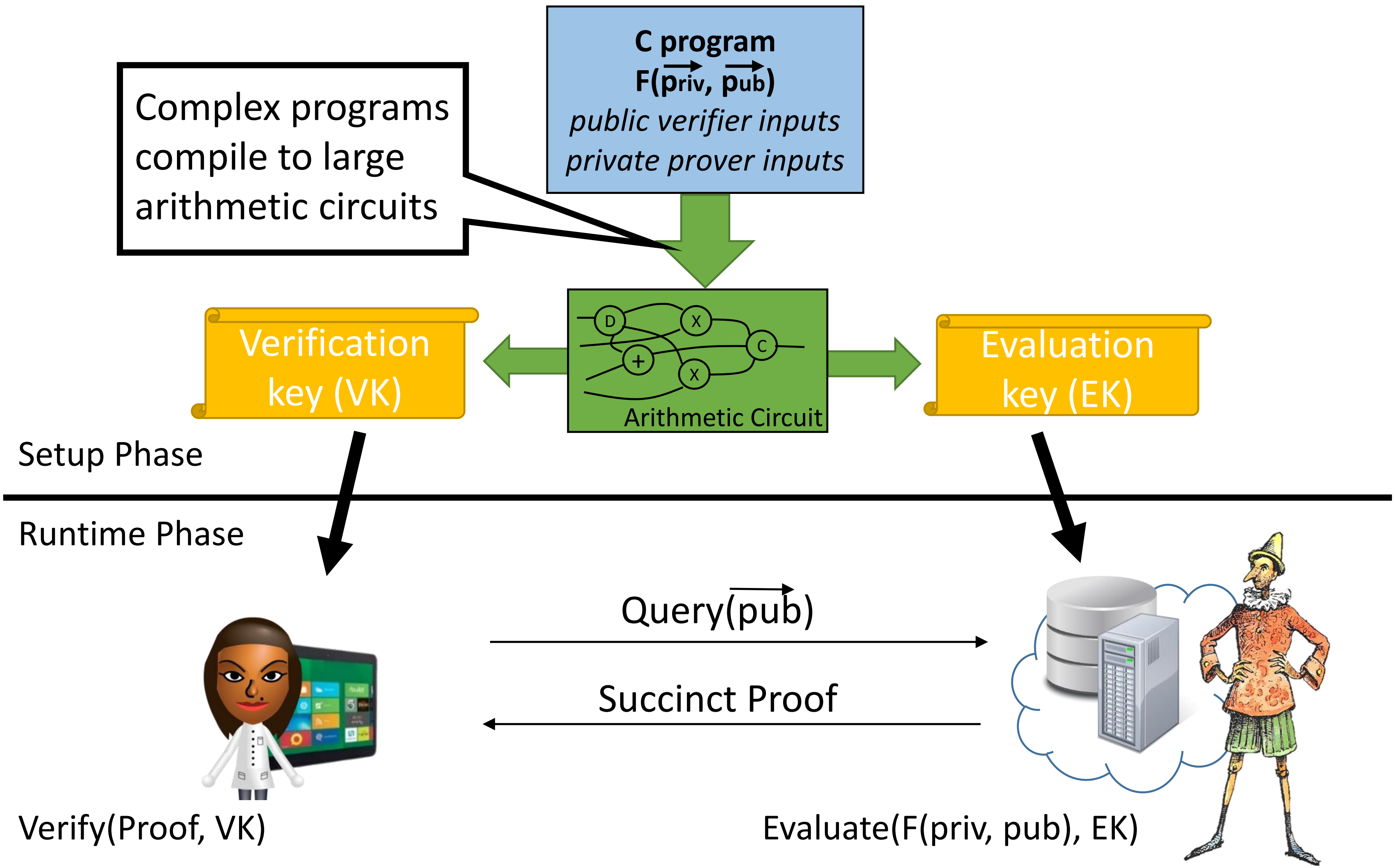


Many anonymous credential systems solve this, but ~0 are used today

Cinderella: Main Idea



Computation Outsourcing with Pinocchio



[CRYPTO '10] [EuroCrypt'13] [IEEE S&P '13] [IEEE S&P '15]

Cinderella: Contributions

- A compiler from high-level validation policy templates to Pinocchio-optimized certificate validators
- Pinocchio-optimized libraries for hashing and RSA-PKCS#1 signature validation
- Several TLS validation policies based on concrete templates and additional evidence (OCSP)
  - Integrated with OpenSSL
  - Tested on real certificate chains
- e-Voting support based on Helios with Estonian ID cards

Benefits and Caveats

- **Practicality:** Compatible with existing PKI and certificates
- Ensures **uniform application of the validation policy** but allows **flexible issuance policies**
- **Anonymity**: Complete control over disclosure of certificate contents
- Less exposure of long-term private keys through weak algorithms

- Computationally expensive
- Initial agreement on the validation policy
- Reliance on security of verified computation system
  - Exotic crypto assumption
  - Trusted key generation
- Does not solve key management (one more layer to manage)

Compiling Certificate Templates

```
seq {seq {
  # Version
  tag<0>: const<2L>;
  # Serial Number
  var<int, serial, 10, 20>;
  # Signature Algorithm
  seq {
const<O1.2.840.113549.1.1.5>
; const<null>; };

  # Issuer
  seq { set { seq {
   const<O2.5.4.10>;
const<printable:"AlphaSSL">;
};};set { seq { const<O2.5.4.3>;
const<printable:"AlphaSSL CA -
G2">; }; };
  };
```

```
# Validity Period
  seq {
   var<date, notbefore, 13, 13>;
   var<date, notafter, 13, 13>;
  };

  # Subject
  seq {
   varlist<subject, 2, 4>;
   set {
    seq {
     var<oid, subjectoid, 3, 10>;
     var<x500, subjectval, 2, 31>;
    };
   };
  };
[…]
```

Variables

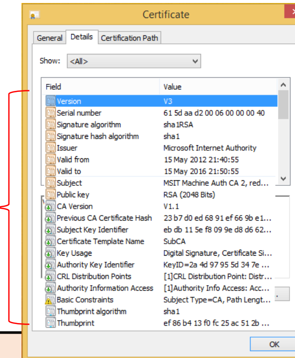
Variable lists

Constants

Template

Template compiler

Private inputs



Untrusted Native Parser
Parse certificate
Generate Prover Inputs

C/QAP verifier
Concatenate compile-time constants and run-time vars
Compute running hash

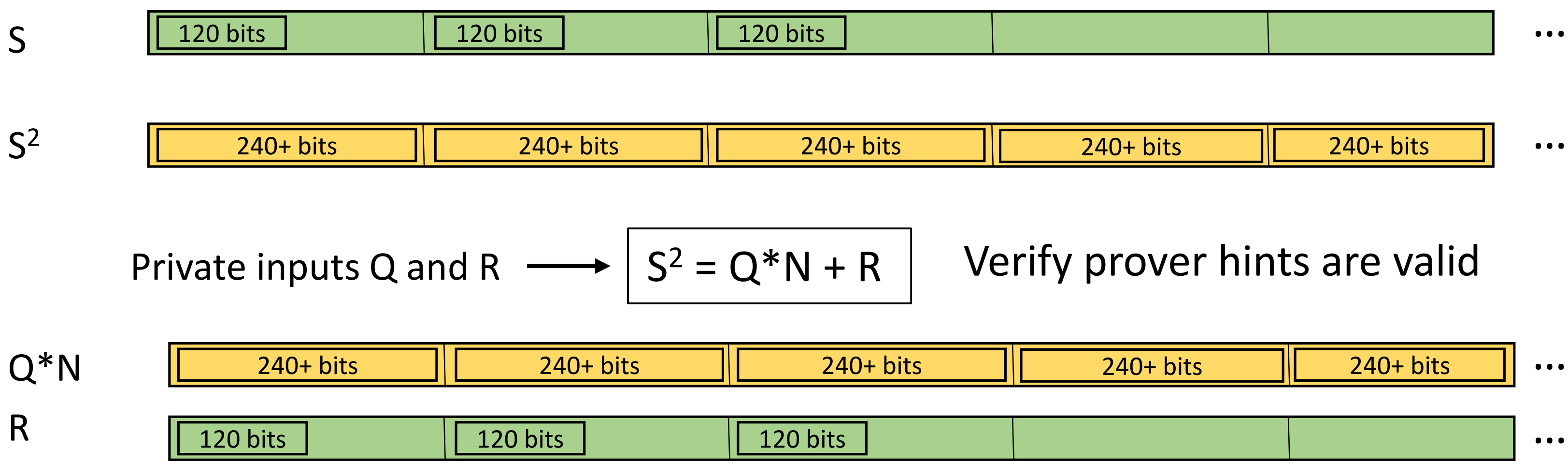
Verifying PKCS#1 RSA Signatures

S ^ e mod N = 1ffffffffff[…]ffffffkkkkk[…]kkkkkkyyyyyyyyyyyyyyyyyyy

S ^ e = S (((S ^ 2) ^ 2) …

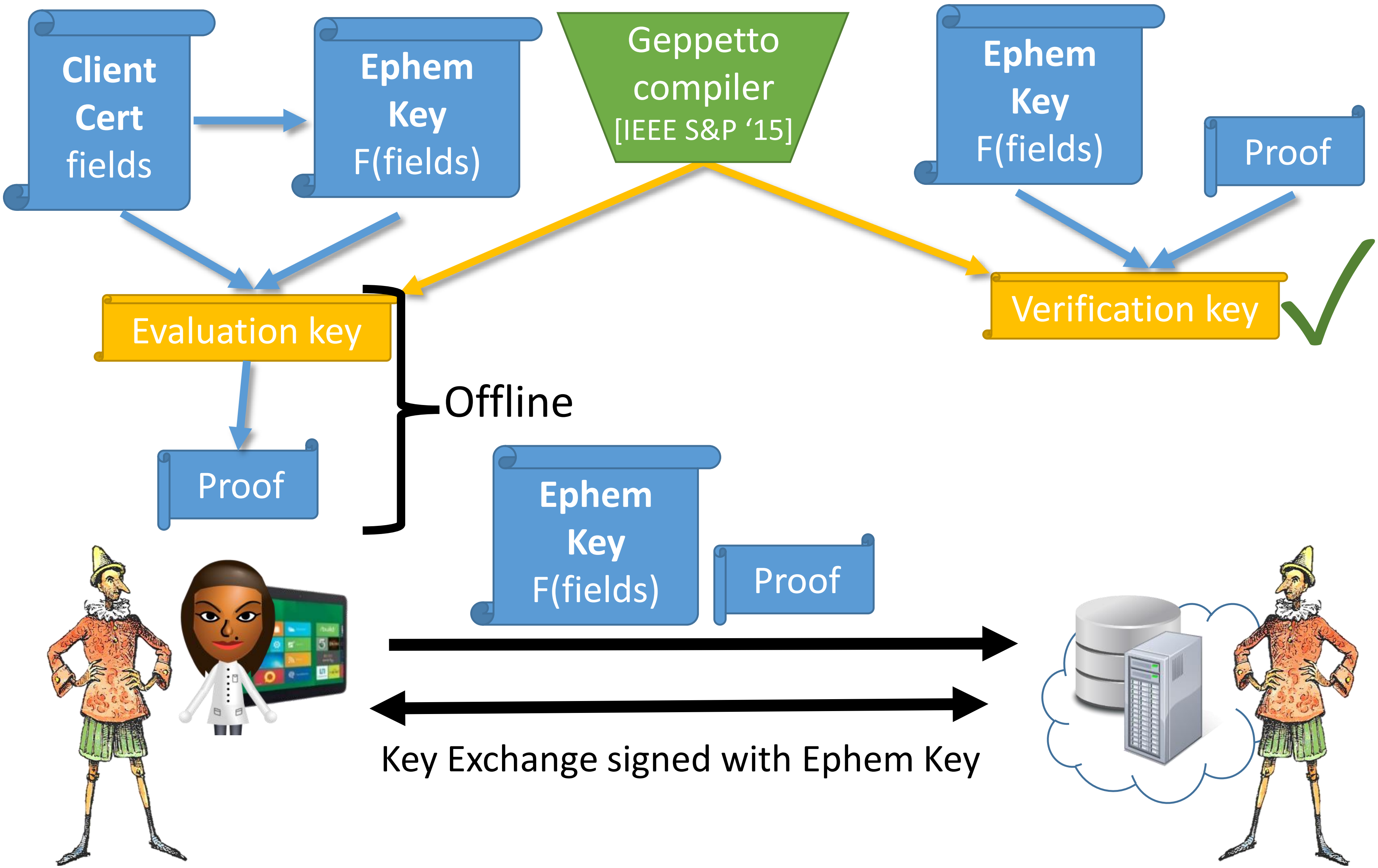
Hash (computed before)

Assume fixed e = 65537 = 2^{16} + 1

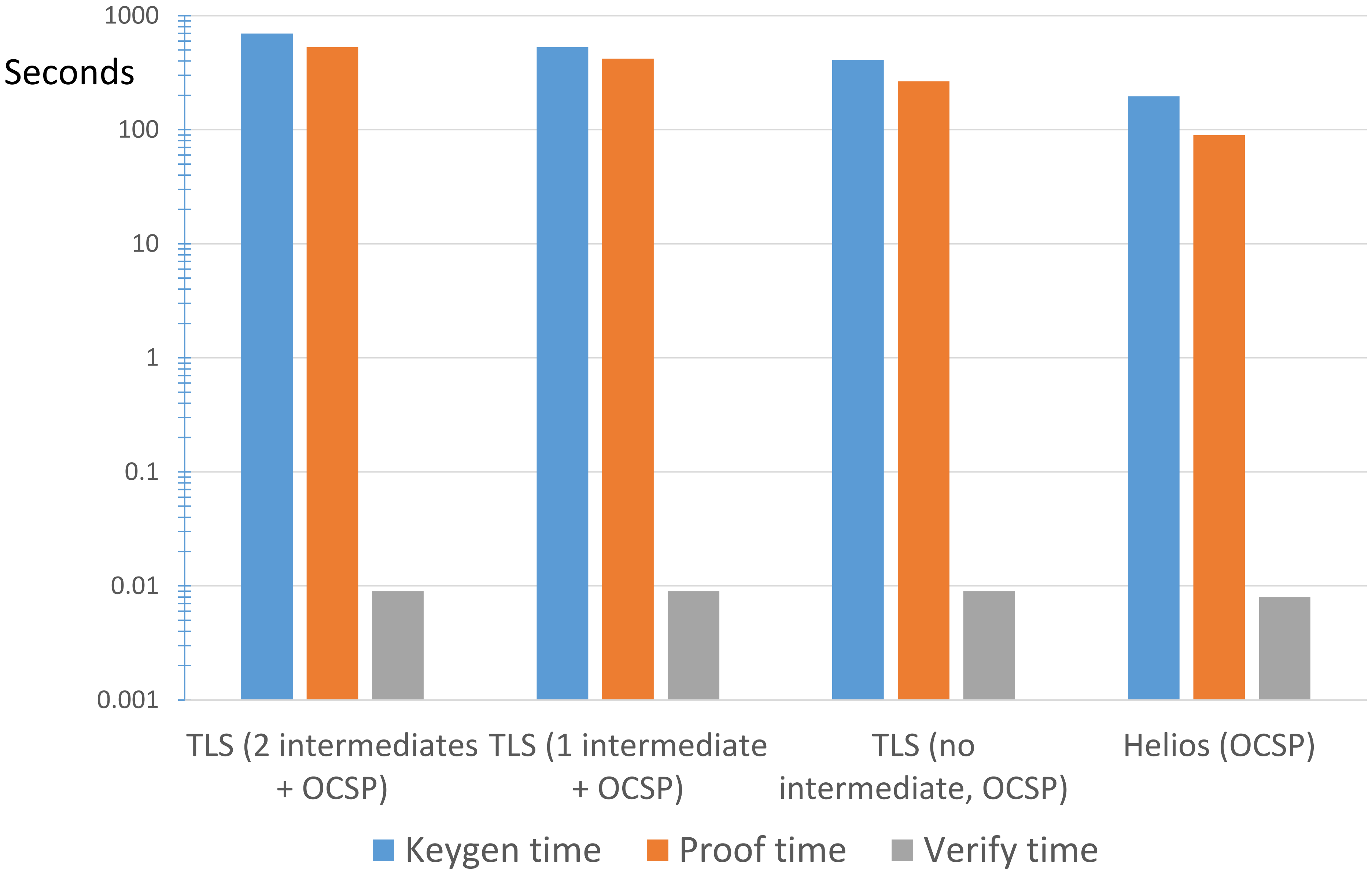


$\boxed{S \leftarrow R}$

Application: TLS Client Authentication



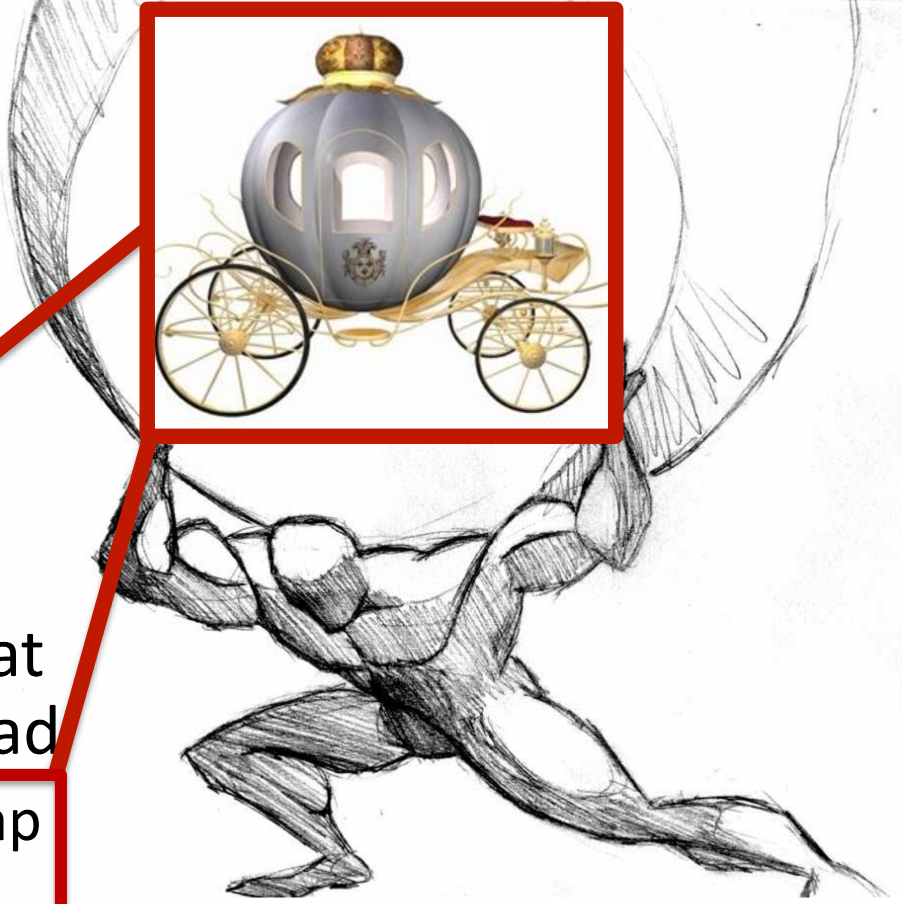
Application evaluation



Cinderella Summary

- One of the first practical applications of verifiable computing
- We achieve privacy and integrity for X.509 authentication
- No change to PKI or to protocols
- Working prototype for TLS and Helios

Coping with Prover Overhead



1. Leverage zero knowledge
   – Example: Bitcoin++
     [Danezis et al. '13] [Ben-Sasson et al. '14] [Kosba et al. '15] [Miller et al. '15]
2. Find (rare?) applications that tolerate substantial overhead
   – Original computation is cheap or infrequent
     - Example: Fair exchange of digital goods [Maxwell '16]

   – Integrity benefits outweigh costs
     - Example: Verifiable ASICs [Wahby et al. '15]
3. Innovations in proof generation

Recent Innovations in Proof Generation

- Improve efficiency of popular programming paradigms
  - Ex: Hash-and-Prove [Fiore et al. ‘16]
  - Ex: vSQL [Zhang et al. ‘17]

- Meld SNARKs with interactive proofs
  - Ex: Allspice [Vu et al. ’13], vSQL [Zhang et al. ‘17]

Future Innovations in Proof Generation

- More efficient cryptographic encodings
  - Lattices?
  - Symmetric homomorphic primitives?

- Specialized verifiable computation protocols
  - Ex: ZK verifiable regular expressions

Disruptive Approaches

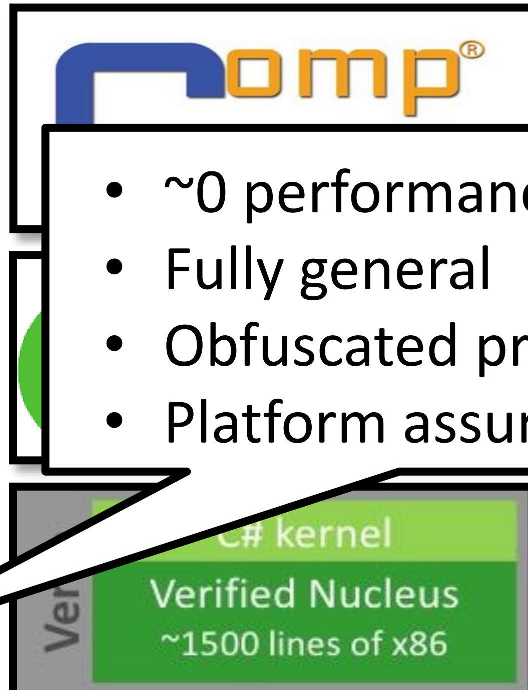
Ubiquitous secure hardware

Trusted Platform Module (TPM)

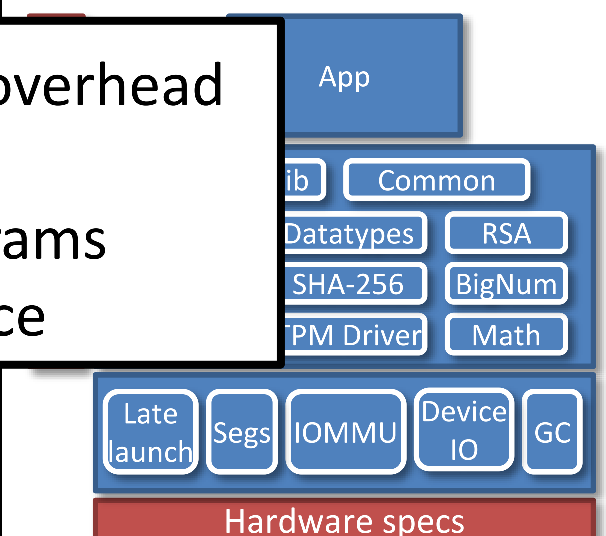
Software Guard Extensions (SGX)

\+

Fully verified software



Ironclad Apps



Secure verifiable computation ?

- ~0 performance overhead
- Fully general
- Obfuscated programs
- Platform assurance

Conclusions

- Despite progress, prover overheads limits usefulness of verifiable computation
- Cinderella circumvents prover overhead to improve the privacy, security, and flexibility of the X.509 PKI
- Secure hardware + verified software may disrupt crypto-only solutions

Thank you!

parno@cmu.edu