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

with the Magic of Verifiable Computation

Antoine Delignat-Lavaud

Cédric Fournet, Markulf Kohlweiss,

Bryan Parno

Microsoft® Research

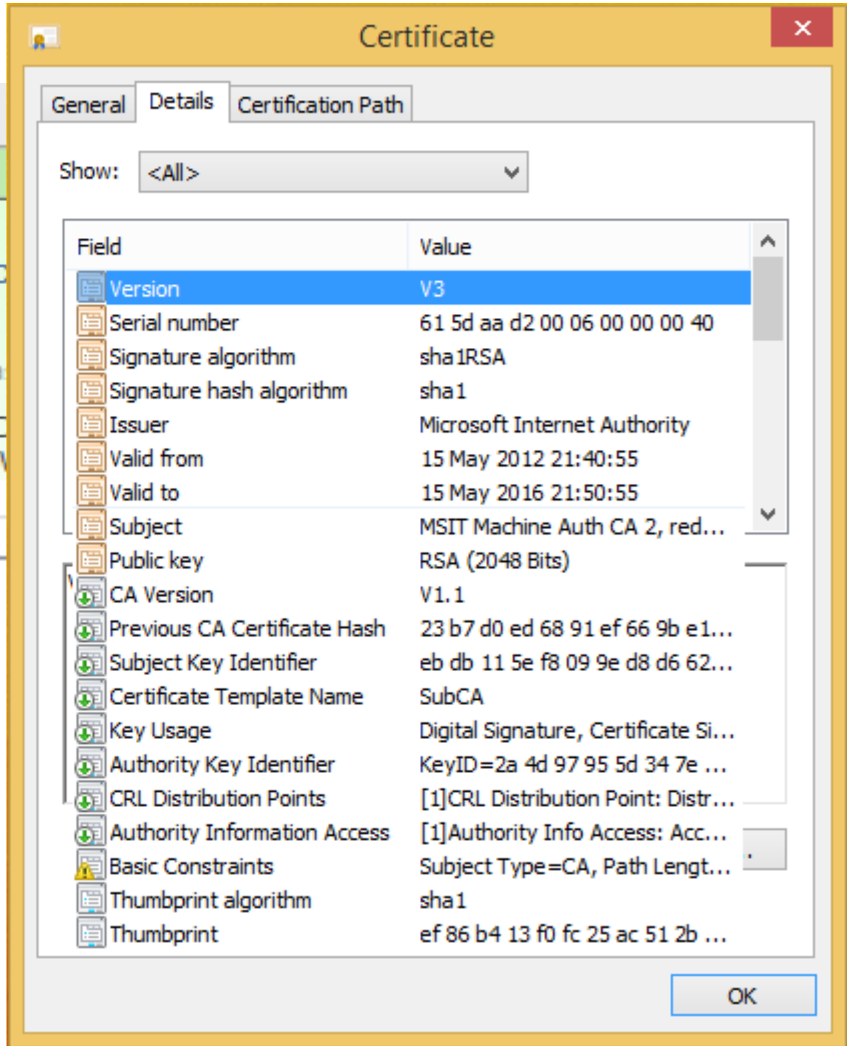
The X.509 Public Key Infrastructure (1988)

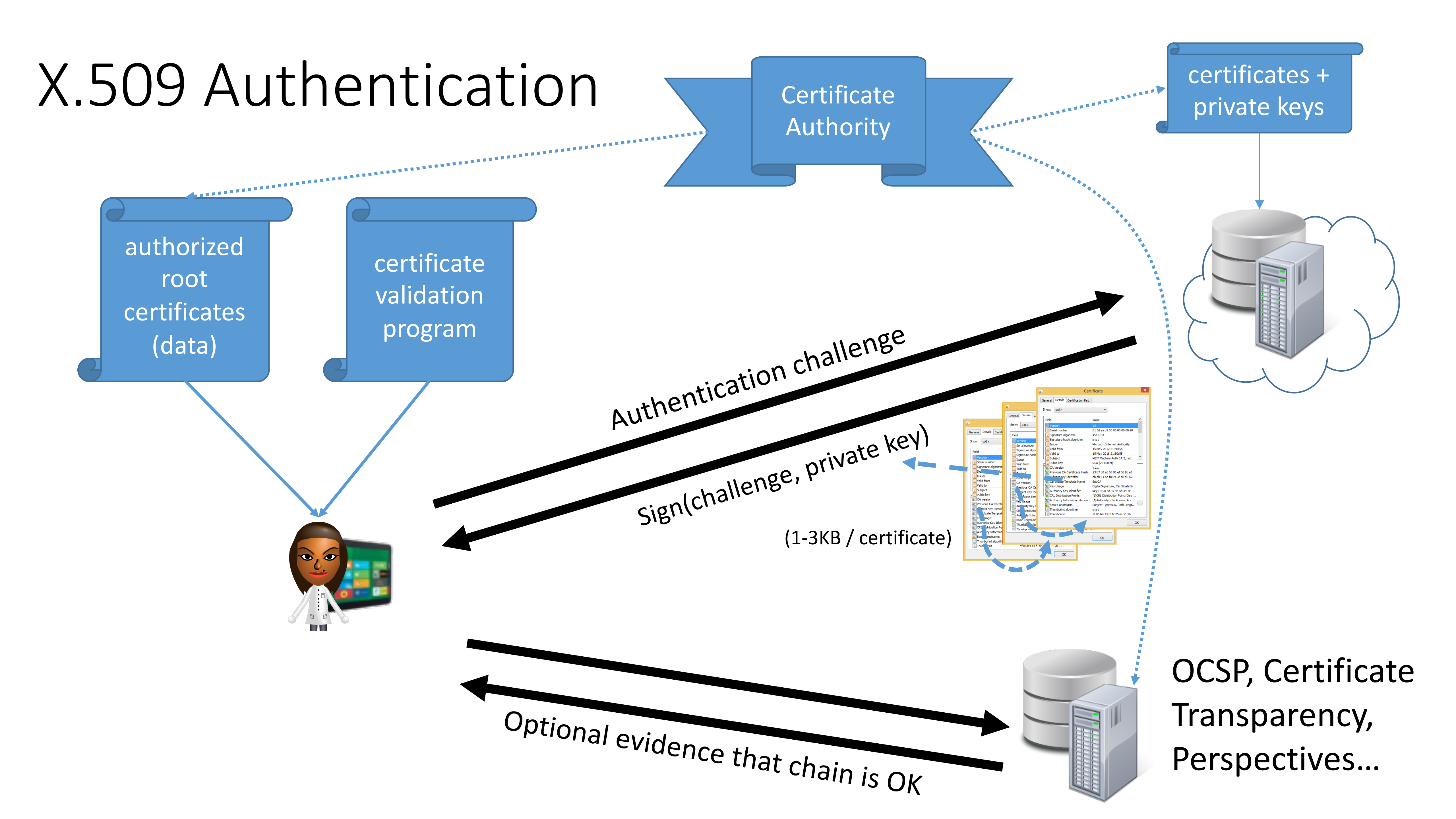
Chain

Endpoint certificate

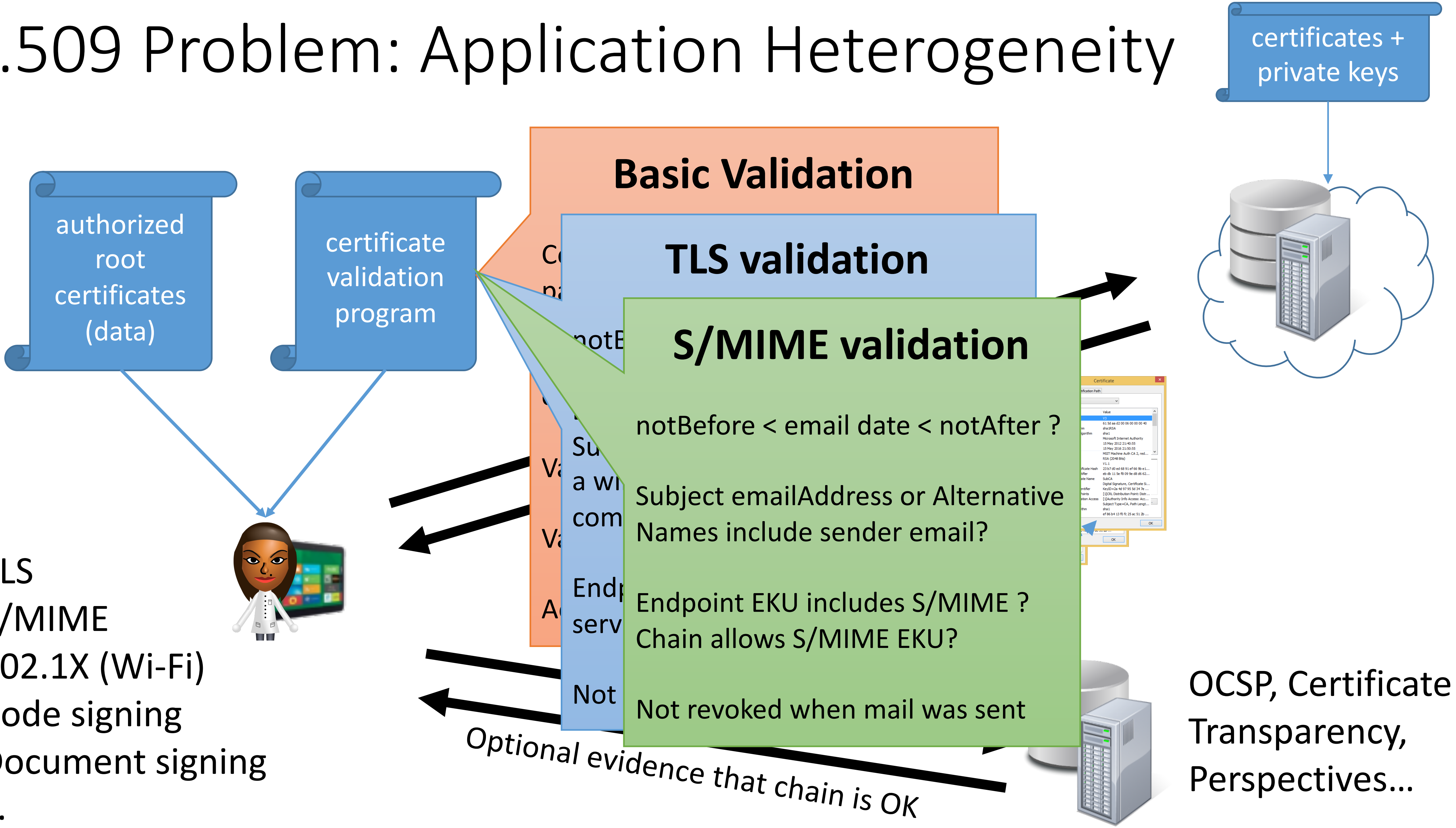
Intermediate Certificate Authority certificate

Root Certification Authority certificate



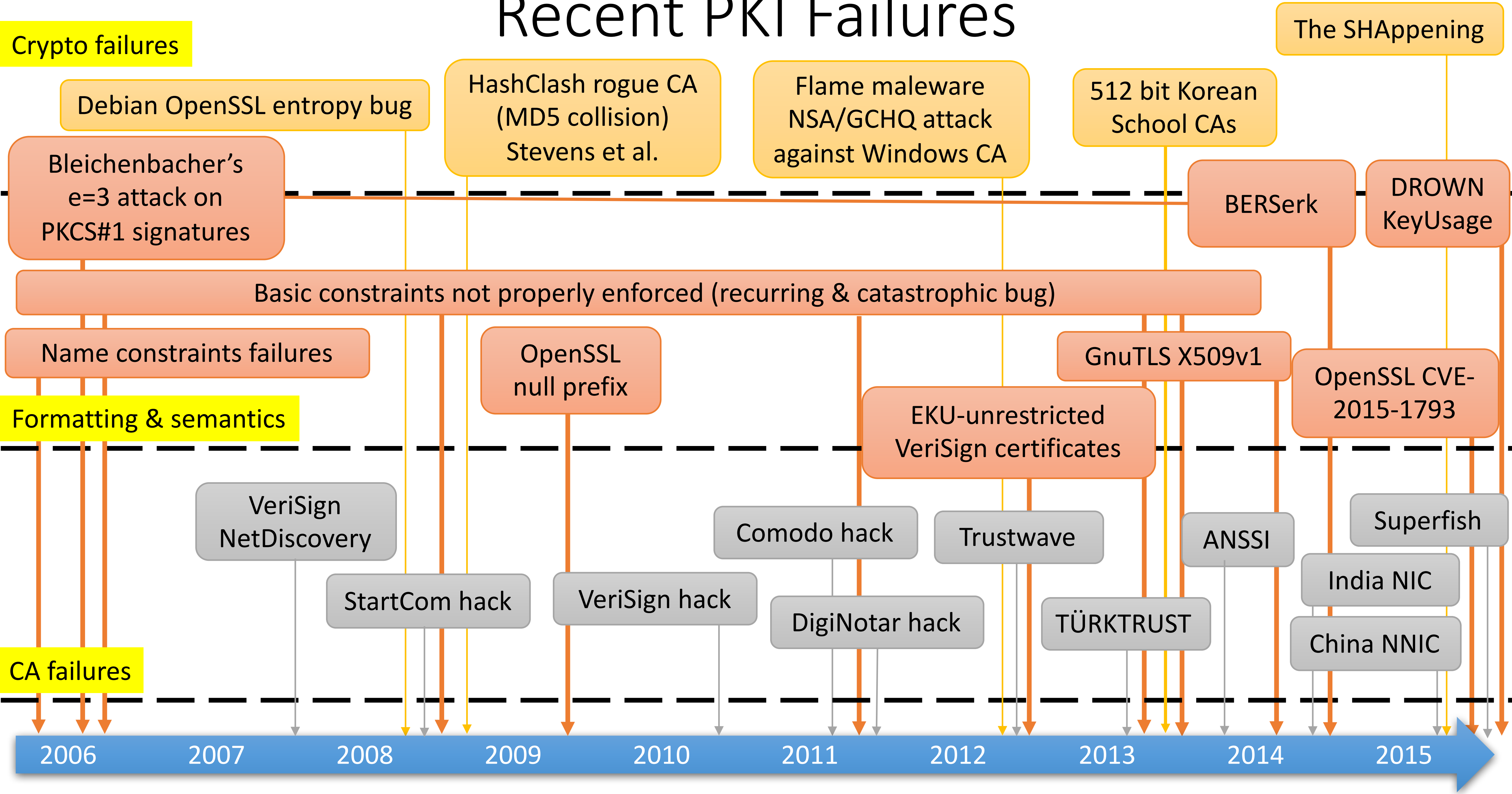

X.509 Authentication
Certificate Authority
certificates + private keys
authorized root certificates (data)
certificate validation program
Authentication challenge
Sign(challenge, private key)
(1-3KB / certificate)
Optional evidence that chain is OK
OCSP, Certificate Transparency, Perspectives…

X.509 Problem: Application Heterogeneity

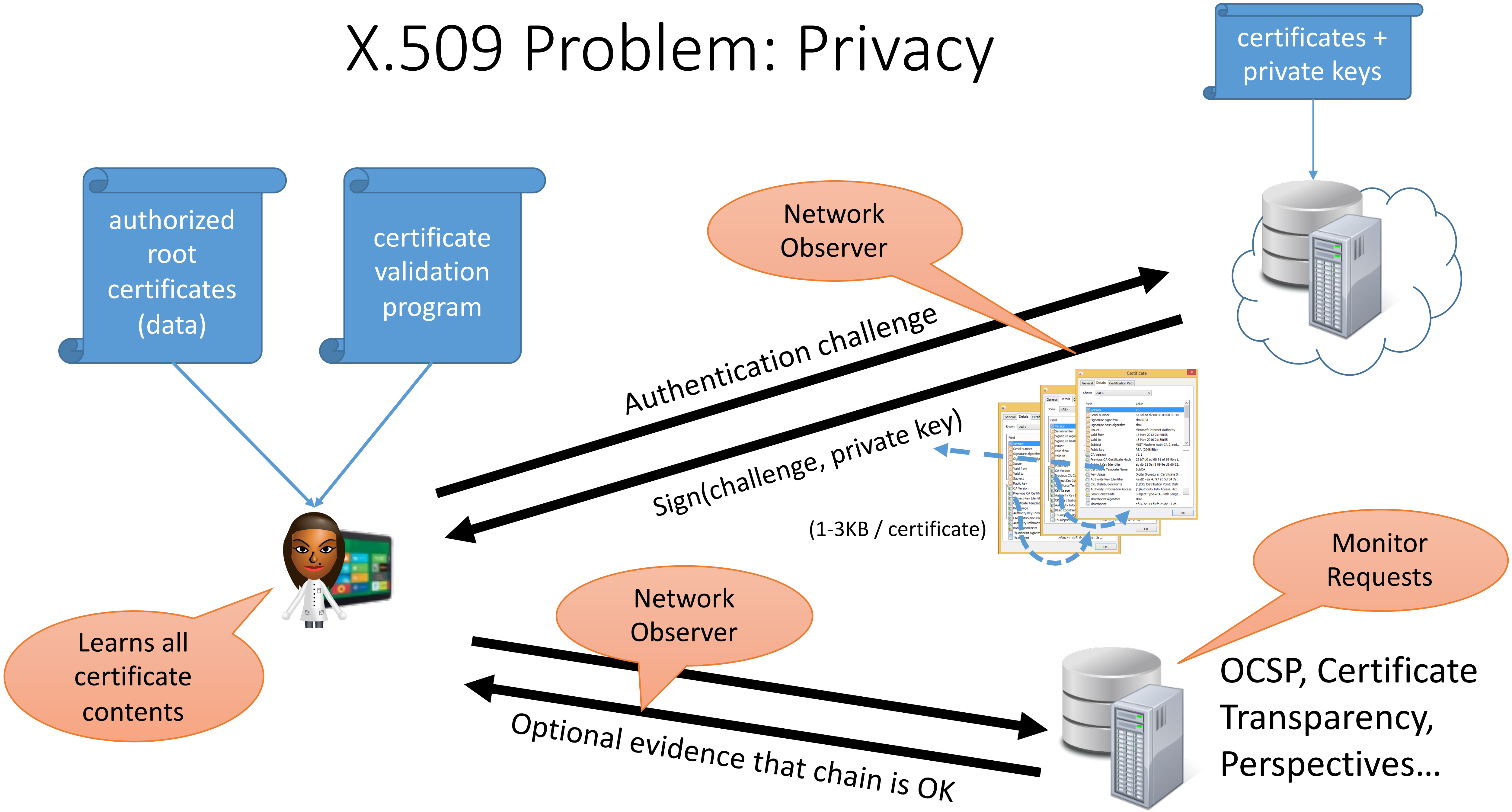


- TLS
- S/MIME
- 802.1X (Wi-Fi)
- Code signing
- Document signing
- …

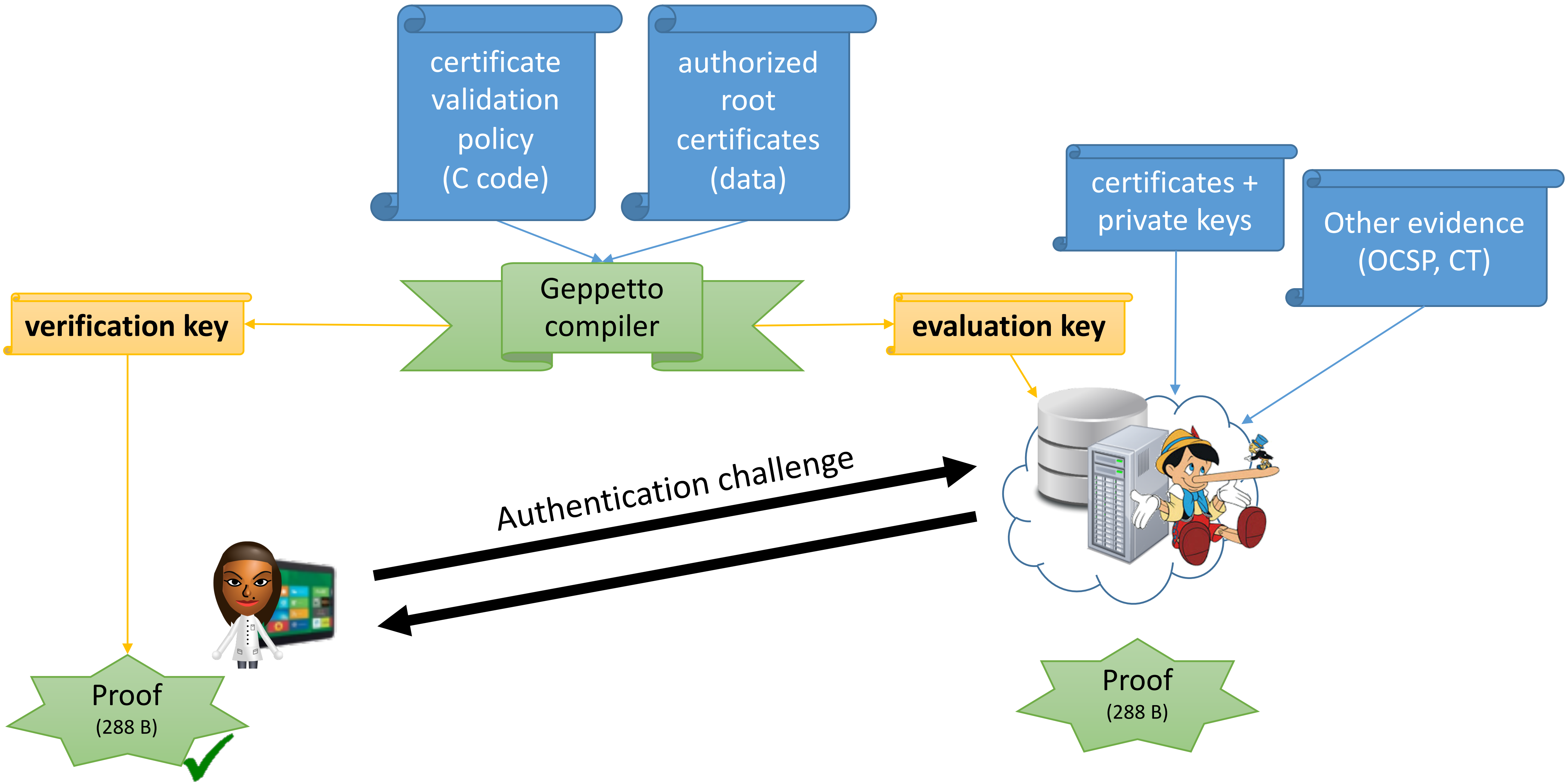
Recent PKI Failures



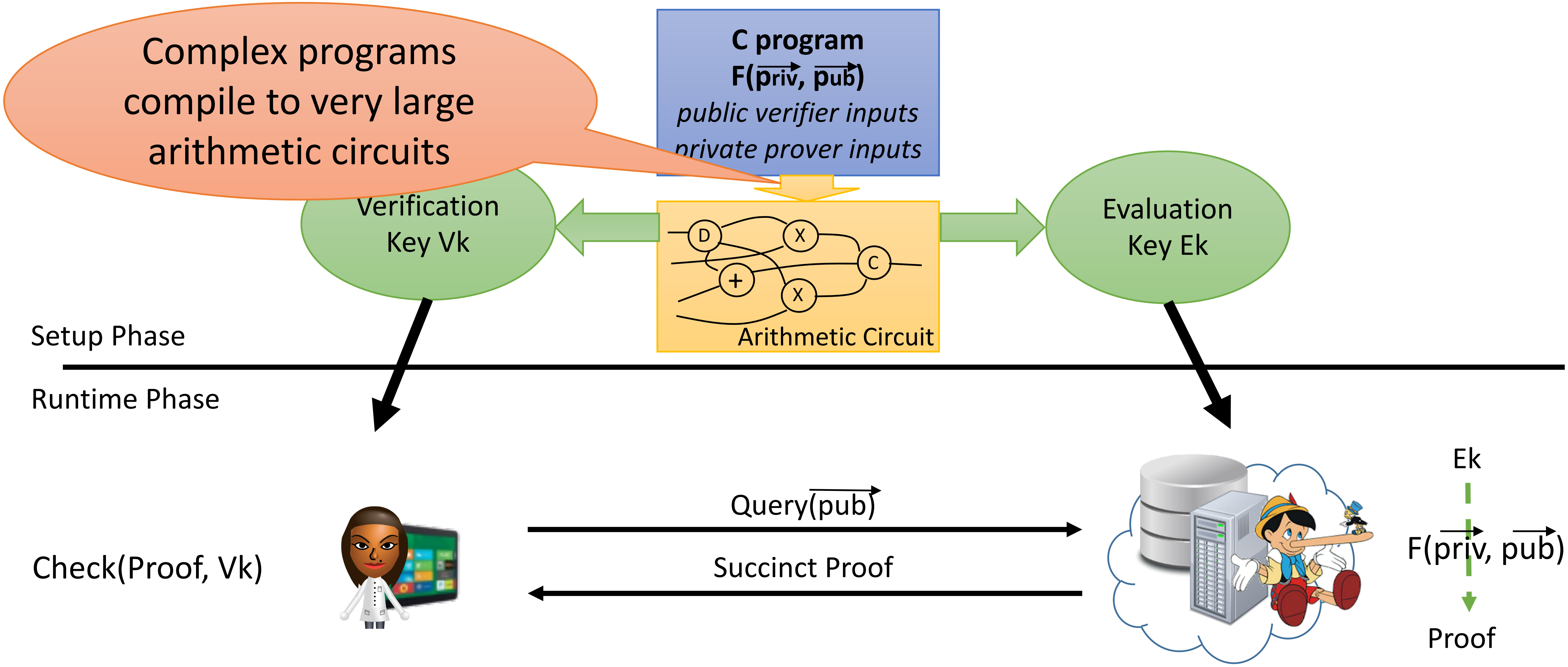
X.509 Problem: Privacy



Cinderella: Main Idea



Computation Outsourcing with Pinocchio



[GGP, CRYPTO'10]; [GGPR, EUROCRYPT'13]; [PGHR; S&P'13]; [CFHKKNBZ; S&P'15]

Cinderella: Contributions

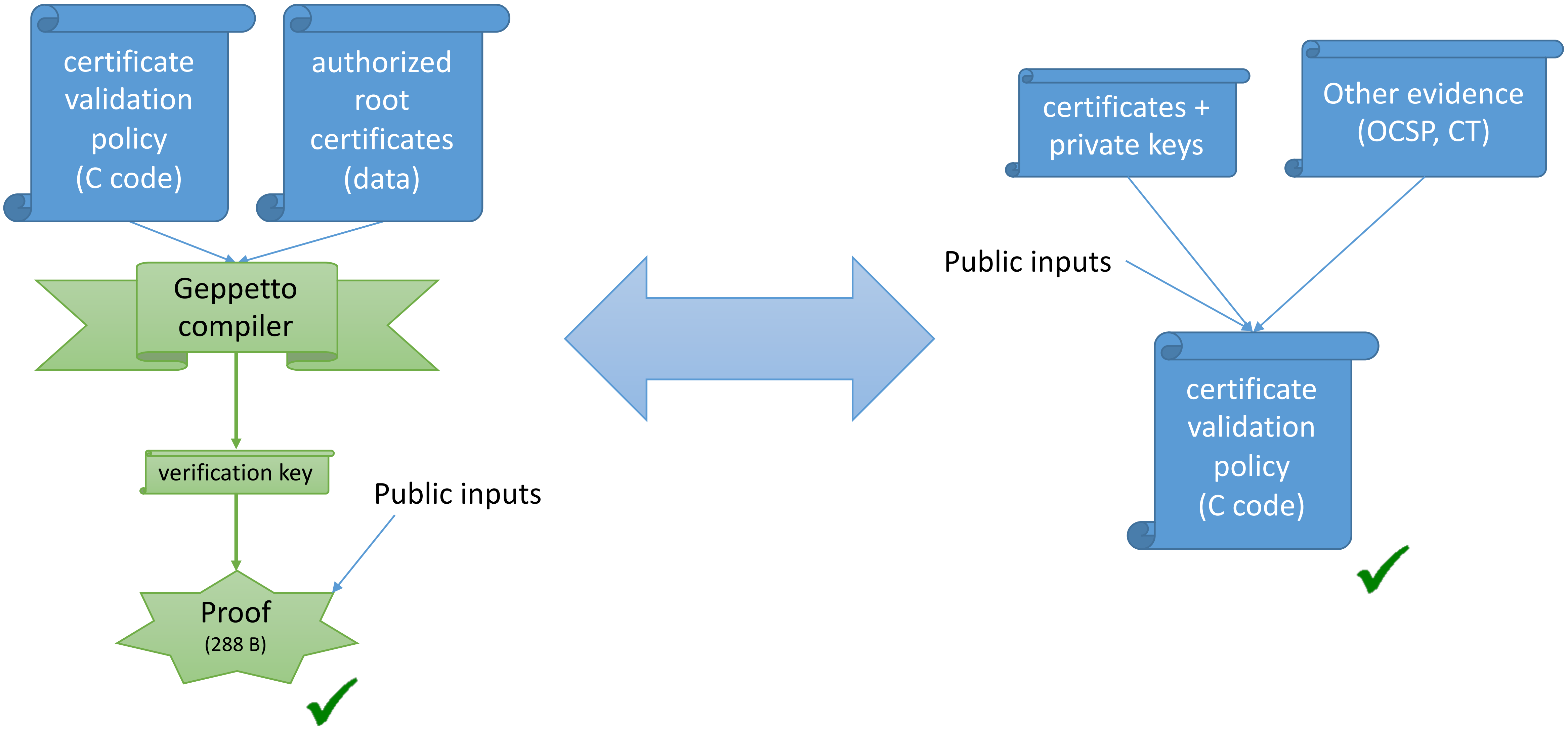
- A compiler from high-level validation policy templates to Pinocchio-optimized certificate validators
- Pinocchio-optimized libraries for hashing and RSA-PKCS#1 signature validation
- Several TLS validation policies based on concrete templates and additional evidence (OCSP), tested on real certificates
- An e-Voting validation policy based on Helios with Estonian ID card

Benefits and Caveats

- **Compatible with existing PKI** and certificates (practicality)
- Ensures **uniform application of the validation policy** but, allows **flexible issuance policies**
- Complete control over disclosure of certificate contents (**anonymity**)
- Less exposure of long-term private key through weak algorithms

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

Cinderella: Soundness



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">; }; };
 };
```

Constants

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

Variables

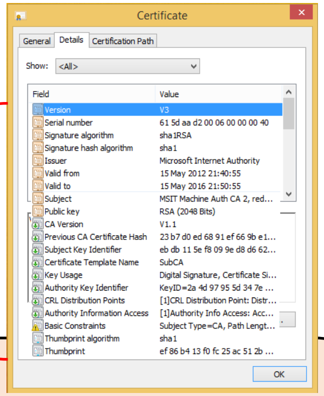
```
 # Subject
 seq {
  varlist<subject, 2, 4>:
  set {
   seq {
    var<oid, subjectoid, 3, 10>;
    var<x500, subjectval, 2, 31>;
   };
  };
 };
[…]
```

Variable lists

Template

Template Verifier 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

Produced Verifier (Fragment)

Variable list

Constants

```
if(in_subject.v[0] > 2) {
  append(&buffer, in_subjectval[2].tag);
  append(&buffer, 0 + LEN(in_subjectval[2]));
  for(i=0; i<31; i++)
   if(i<LEN(in_subjectval[2]))
    append(&buffer, in_subjectval[2].v[i]);
}
```

Variable

Compression

```
if(buffer.cur >= 85)
     reduce(&buffer, &hash);
```

C verifier program

Append(byte)
Add given byte to the hashing buffer

Reduce()
compress one block of buffer, update current hash

Current Hash

Hashing buffer = 2 * hash function block size

Output = hash of ASN.1 formatted certificate contents

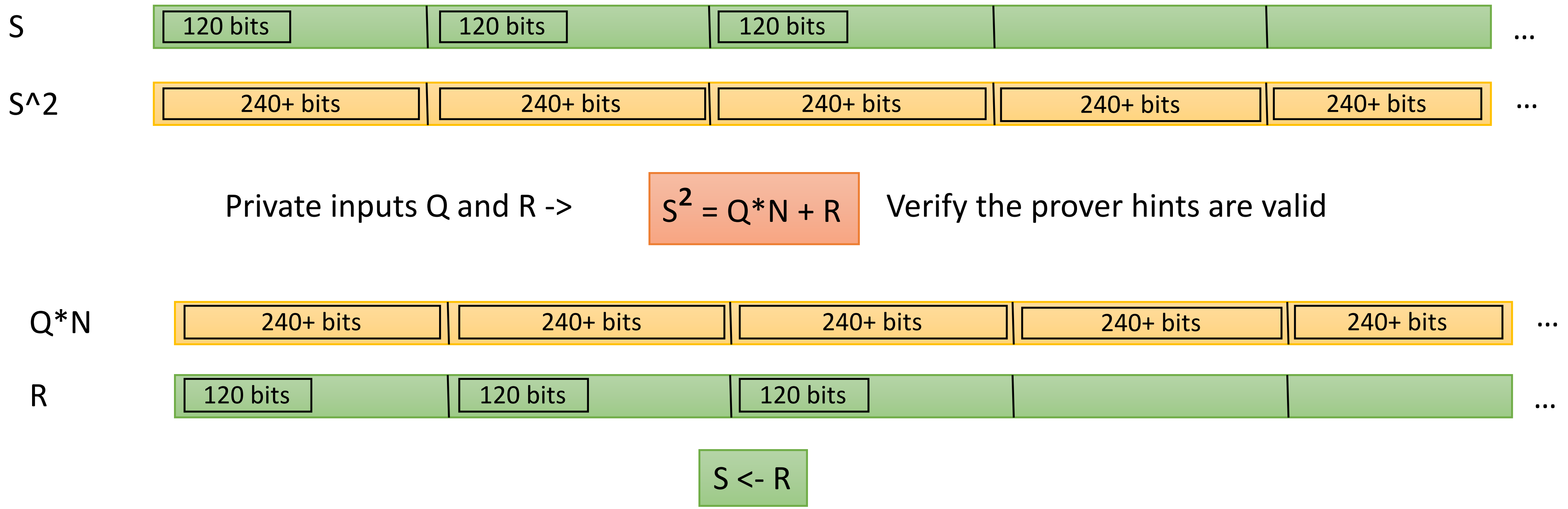
Verifying PKCS#1 RSA Signatures

S ^ e mod N = 1ffffffffff[…]ffffffkkkkk[…]kkkkkk**XXXXXXXXXXXXXXXXXXXXXX**

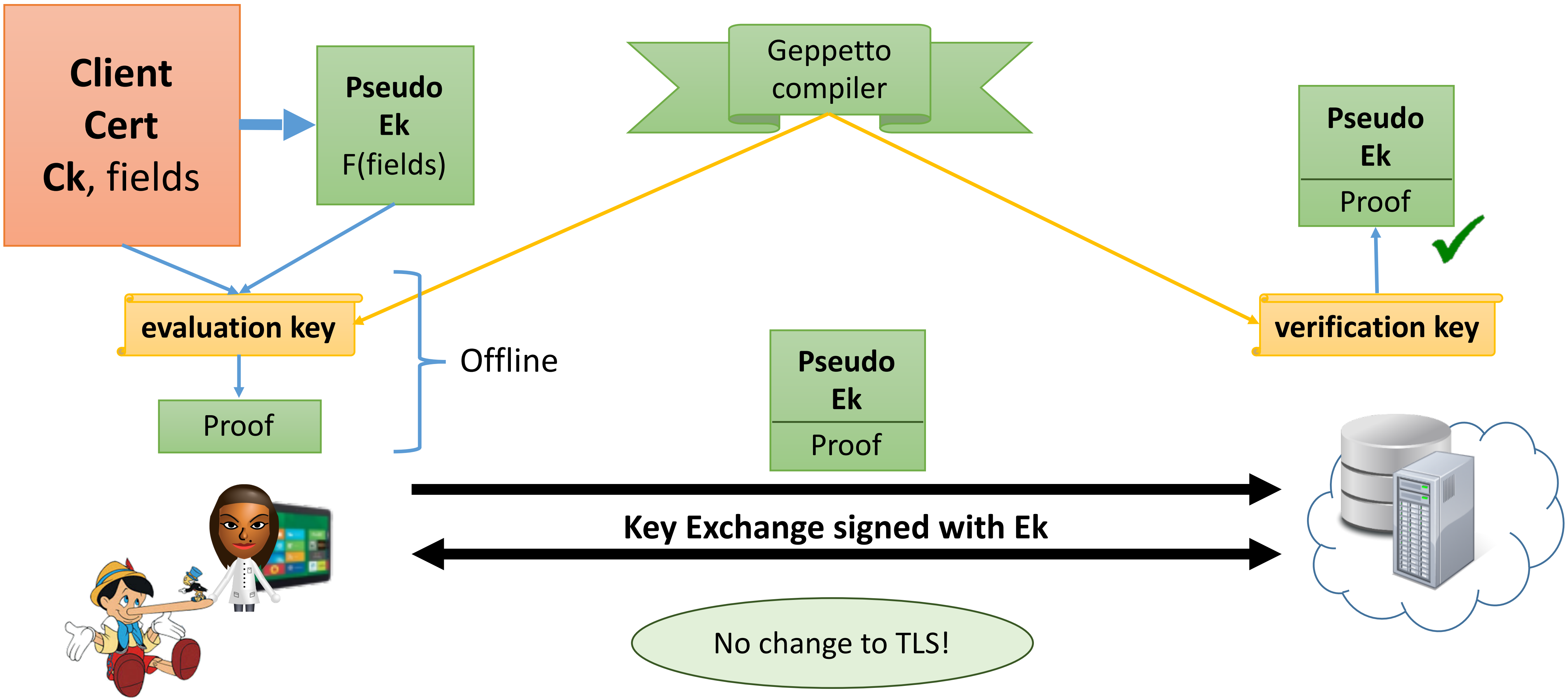
Hash (computed before)

S ^ e = S (((S ^ 2) ^ 2) …

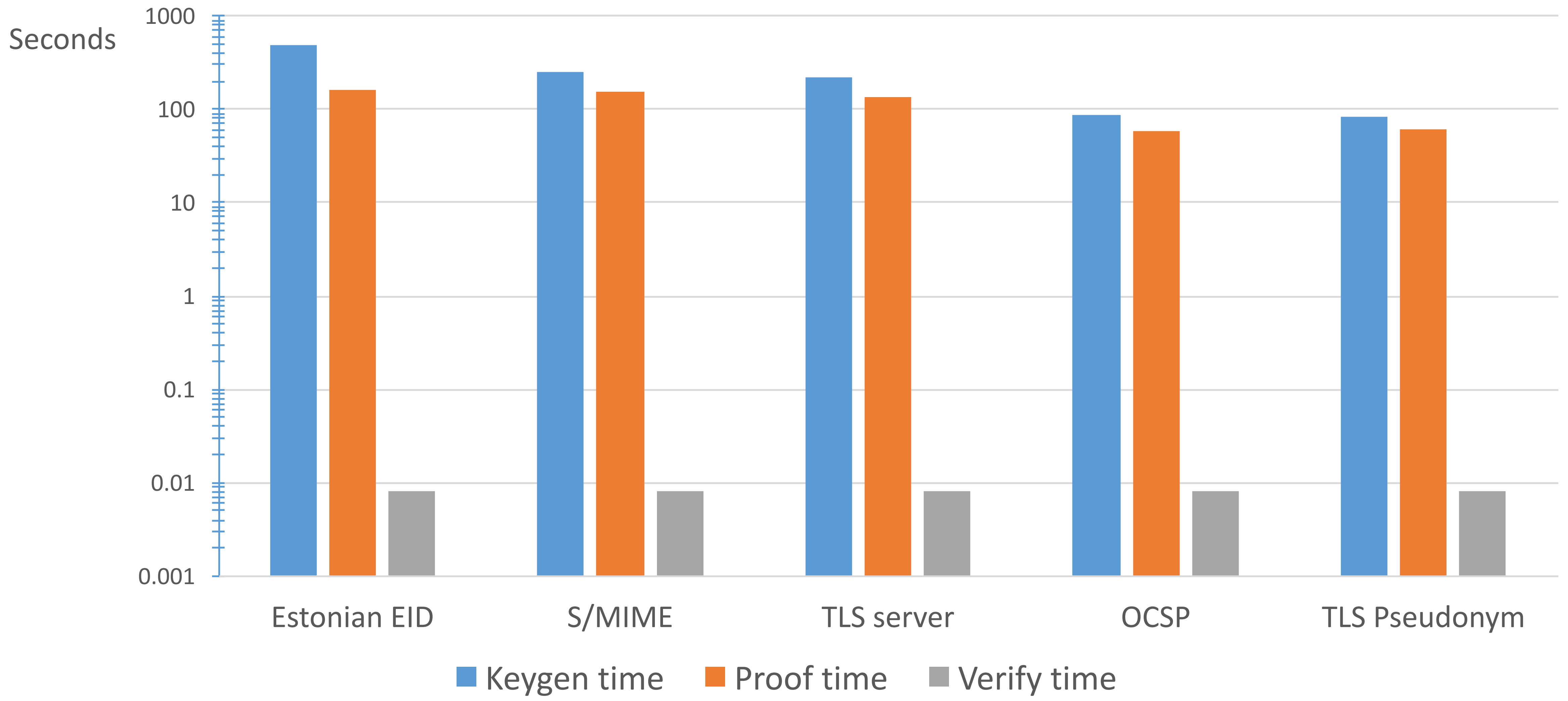
Assume fixed e = 65537 = 2 ^ 16 + 1



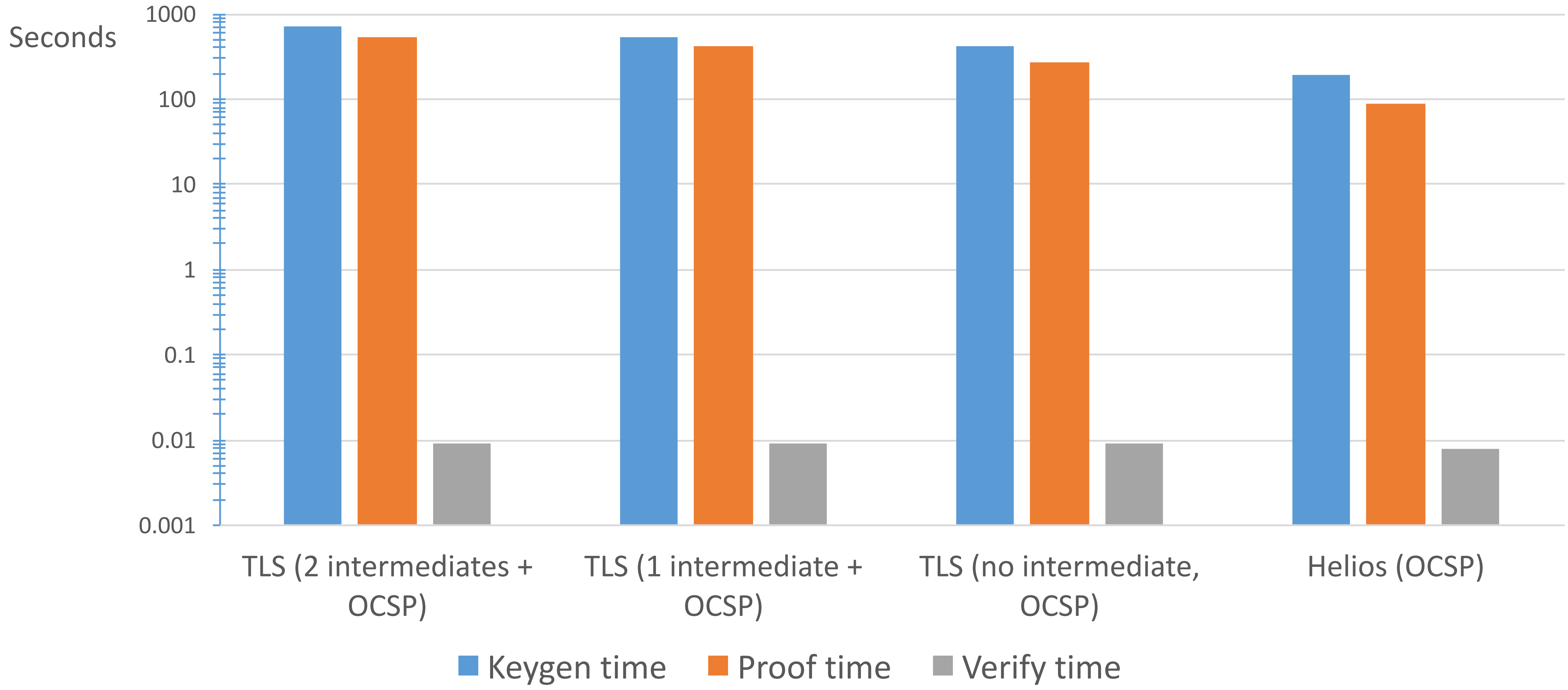
Application: TLS Client (with Offline Signing)



Single Template Evaluation (With Signature)



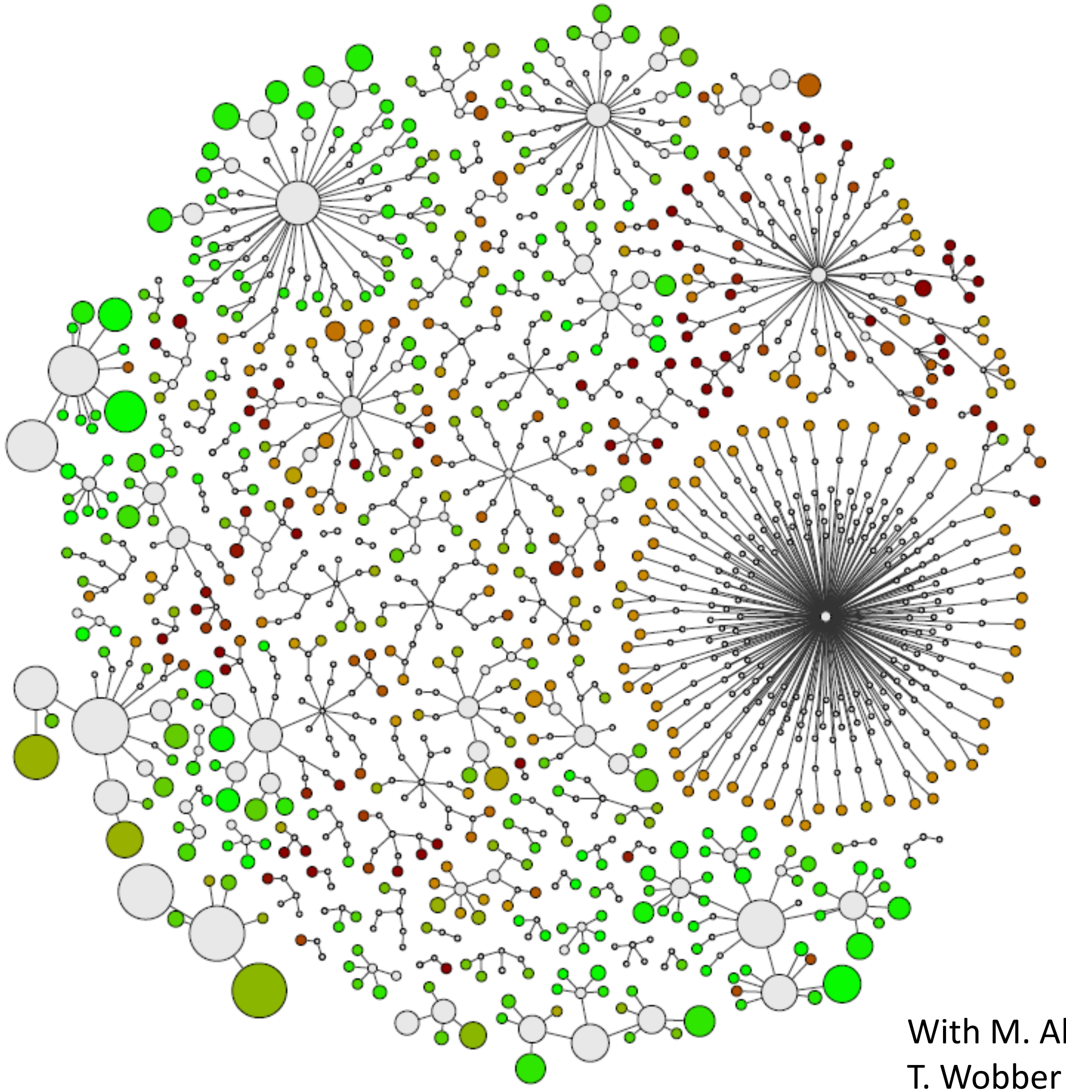
Application evaluation



Conclusions

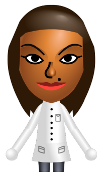
- One of the first practical application of verifiable computing
- We enhance the privacy and integrity of X.509 authentication
- No change to the PKI or to application protocols
- Working prototype for TLS and Helios

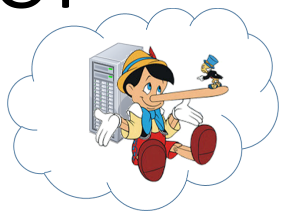
The Internet PKI



With M. Abadi, A. Birrell, I. Mironov, T. Wobber and Y. Xie (NDSS'14)

Core Pinocchio protocol





$KeyGen(R) \longrightarrow EK, VK$

Generate the MultiQAP for R

Pick random s

Compute $EK = \{EK_j\}, \{g^{s^i}\}$

$$EK_j = \begin{Bmatrix} g^{v_k(s)}, g^{w_k(s)}, g^{y_k(s)} \\ g^{\alpha_{j,v} v_k(s)}, g^{\alpha_{j,w} w_k(s)}, g^{\alpha_{j,y} y_k(s)} \end{Bmatrix}_{k \in I_j}$$

Compute $VK = (g^{d(s)} = g^{\prod_i (s-i)})$

$Commit(EK_j, u_j, o_j) \longrightarrow C_j$

Generate the commitment:

$v^{(j)}(s) = \sum_{k \in I_j} u_k v_k(s) + o_{j,w} d(s)$, similarly for w and y

$C_j = (g^{v(j)}, g^{w(j)}, g^{y(j)}, g^{\alpha_{j,v} v(j)}, g^{\alpha_{j,w} w(j)}, g^{\alpha_{j,y} y(j)})$.

$v(s) = \prod v^{(j)}$ and similarly for w and y

$Prove(EK, \boldsymbol{u}, \boldsymbol{o}) \longrightarrow \pi$

Find $h(x)$ s.t. $h(x) * d(x) = v(x) * w(x) - y(x)$

Compute $g^{h(s)} = \prod \left(g^{s^i}\right)^{h_i}$

Proof is $(g^{v(s)}, g^{w(s)}, g^{y(s)}\ g^{h(s)})$

$Verify(VK_j, C_j)$

$e\left(g^{v(j)}, g_j^{\alpha_{j,v}}\right) = e(g^{\alpha_{j,v} v(j)}, g)$

and similarly for w and y

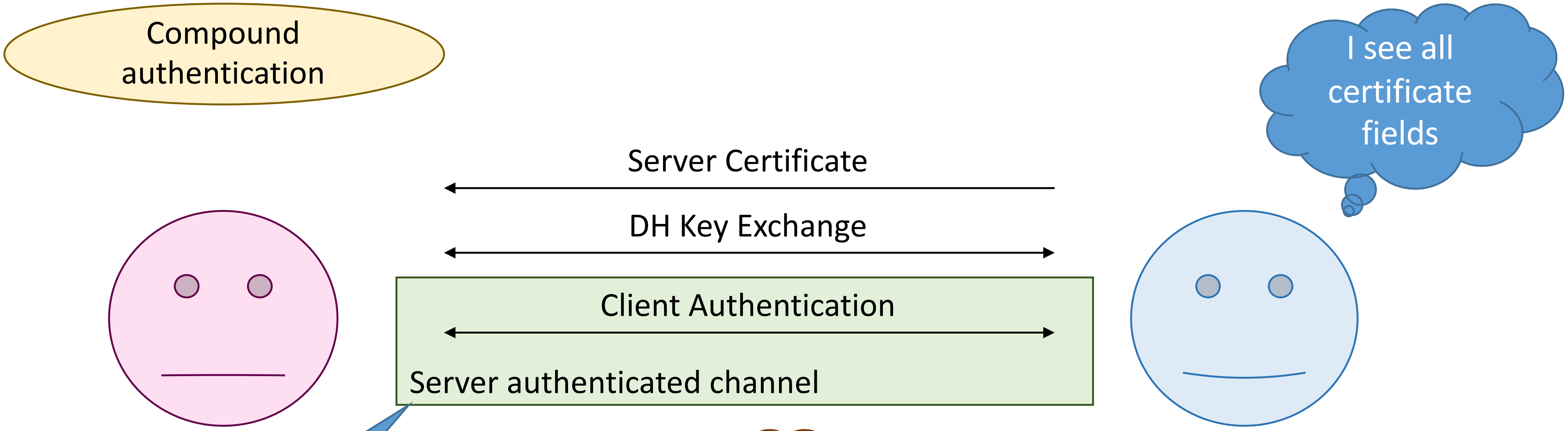
$Verify(VK, \boldsymbol{C}, \pi) \longrightarrow$ {Yes, No}

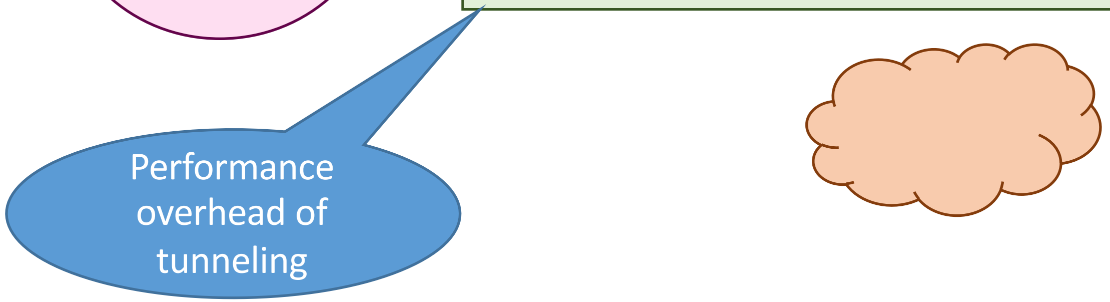
$$\frac{e(g^{v(s)}, g^{w(s)})}{e(g^{y(s)}, g)} \stackrel{?}{=} e(g^{h(s)}, g^{d(s)})$$

$e(\cdot,\cdot)$ is a pairing:

$e(g^p, g^q) = e(g, g)^{pq}$

Workaround: Tunneling



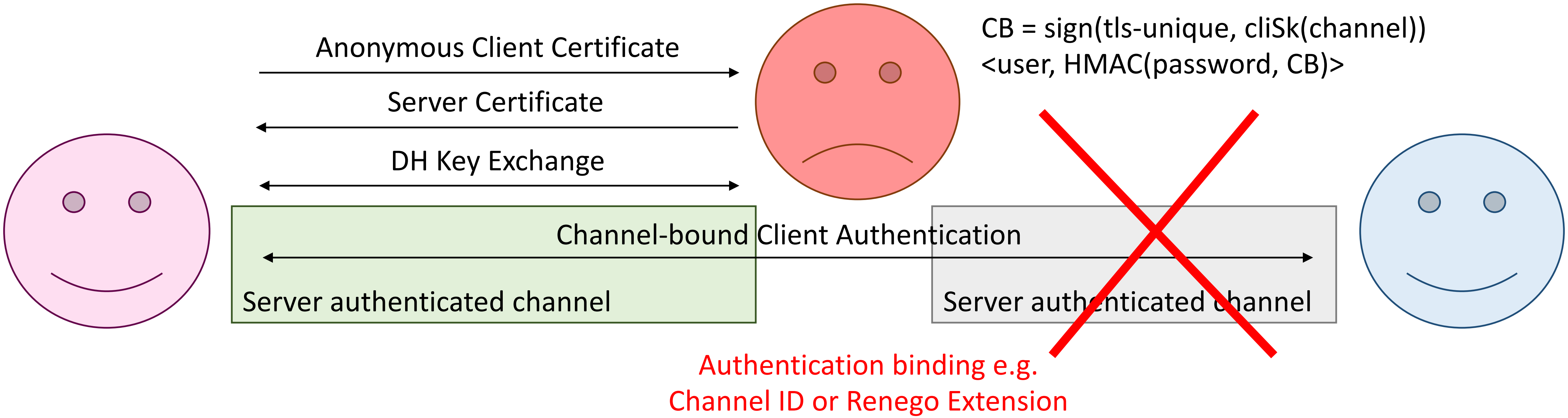


- TLS Renegotiation
- TLS 1.3 Handshake Encryption
- Server still sees all contents
- Not always possible (S/MIME, code and document signing)

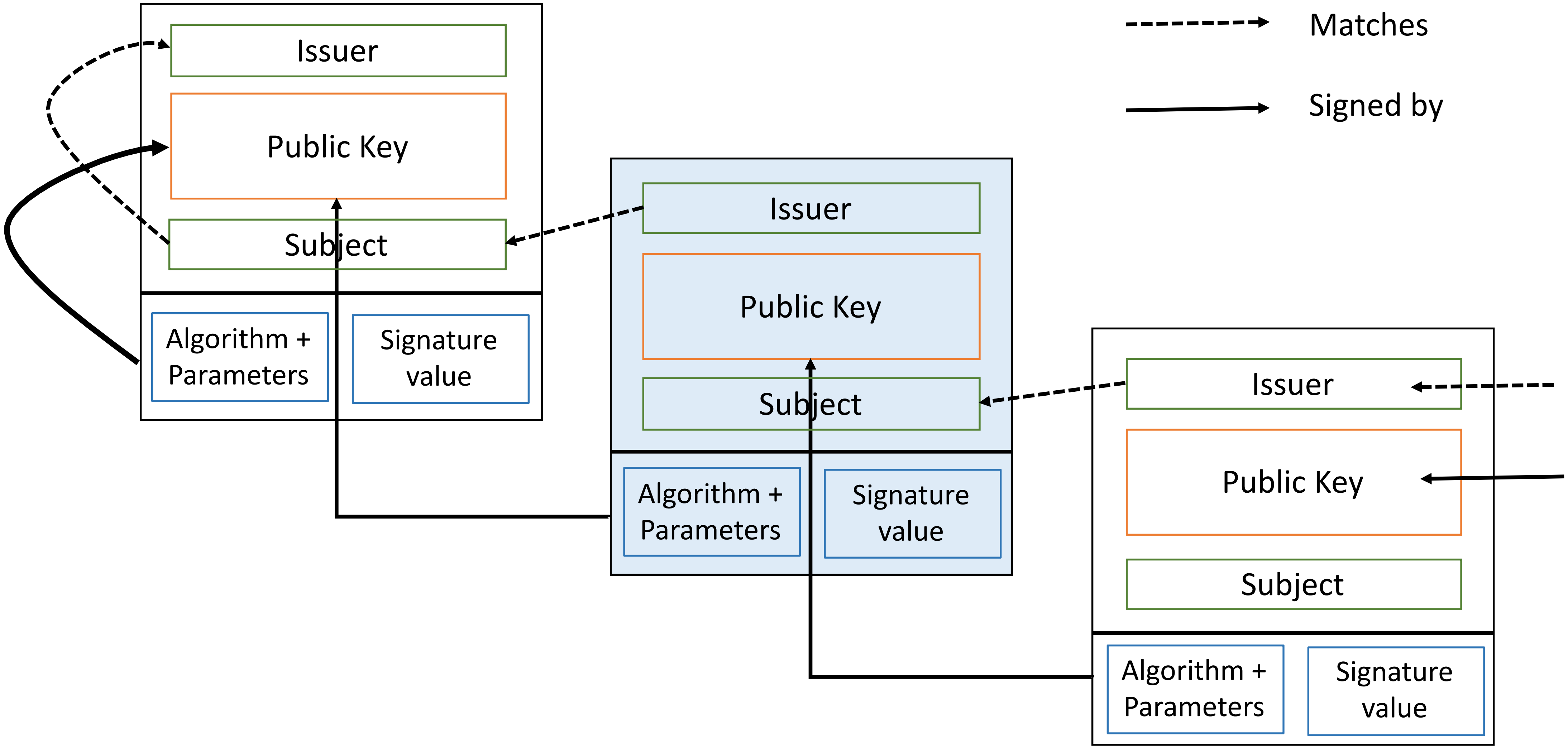
Usability and Privacy of PKI Authentication

Current Privacy Approach

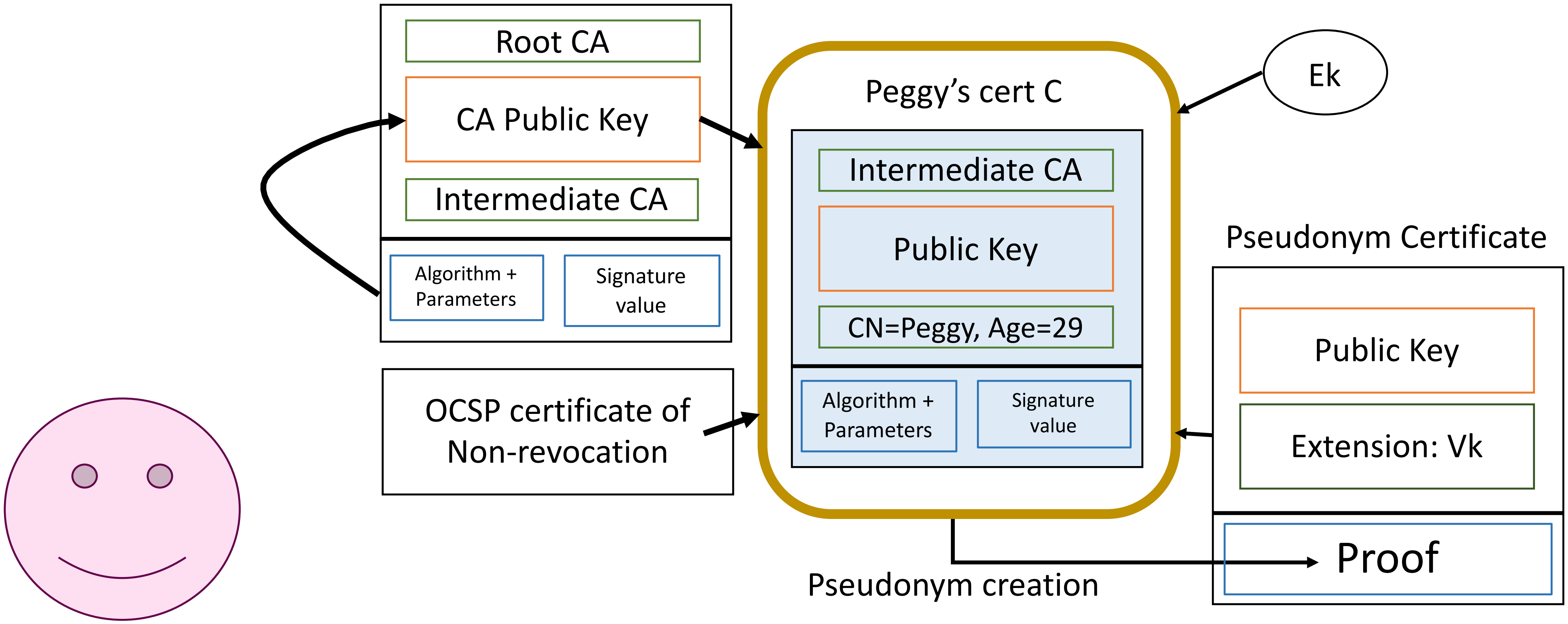
- User Unfriendly
- Complex
- Key Compromise Impersonation attacks



The Internet PKI



Deployment: X.509 Signature Scheme



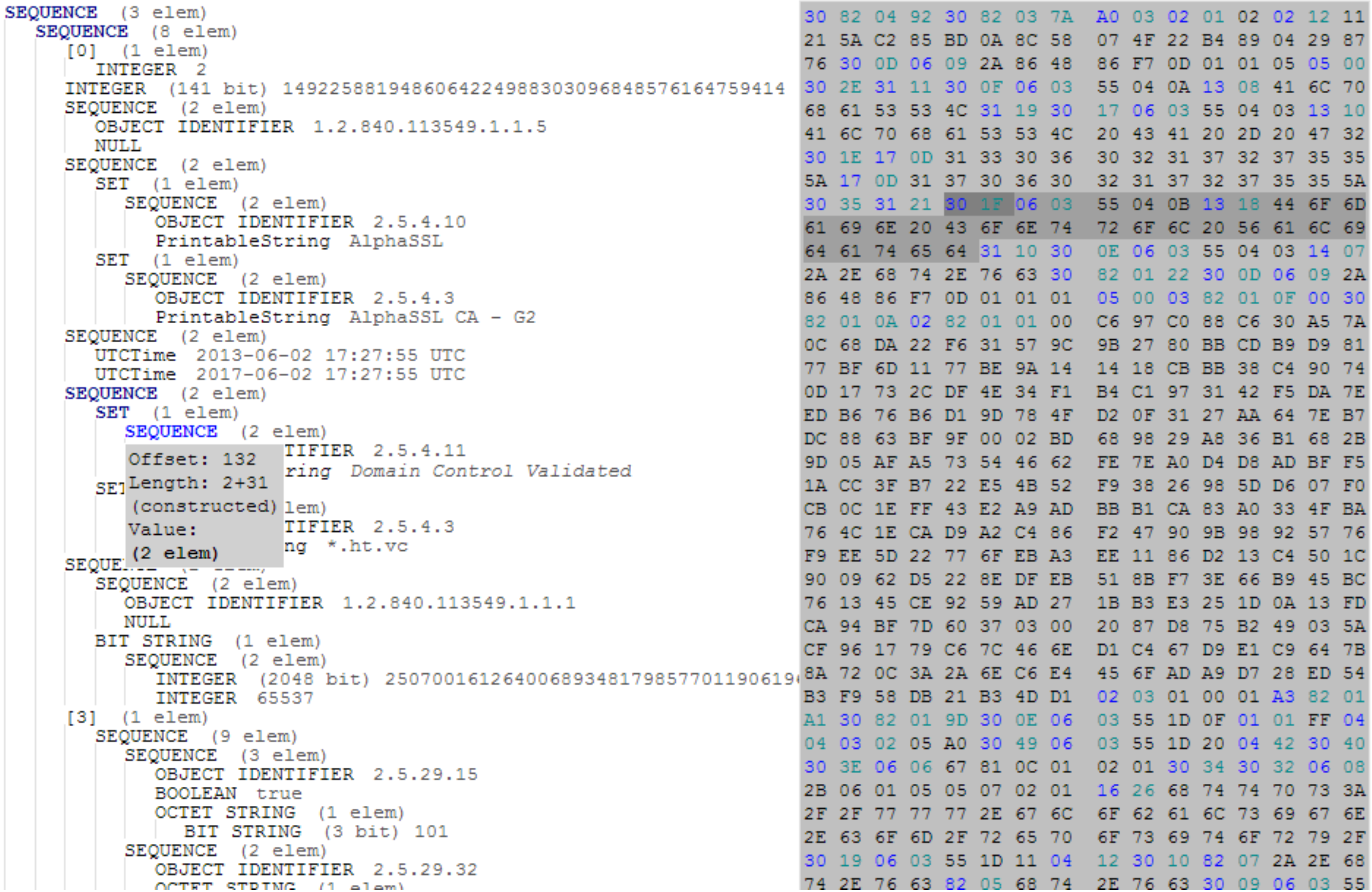
ASN.1

Binary encoding standard

Ancient (1984)

<Tag, Length, Value>

Distinguished rules (DER): unique serialization



Checking RSA Signatures

Assume fixed e = 65537 = 2 ^ 16 + 1

```
for(i=0; i < 17; i++)
{
 if(i<16) big_square(inputs[3], inputs[2]);
 else big_mul(inputs[3], inputs[2], inputs[1]);

 big_mul(inputs[4], inputs[0], quotients[i]);
 big_sub(inputs[5], inputs[3], inputs[4], compl);

 if(!check_eqmod(inputs[5], residues[i], carries[i]))
  return false;

 big_copy(inputs[2], residues[i]);
}
```