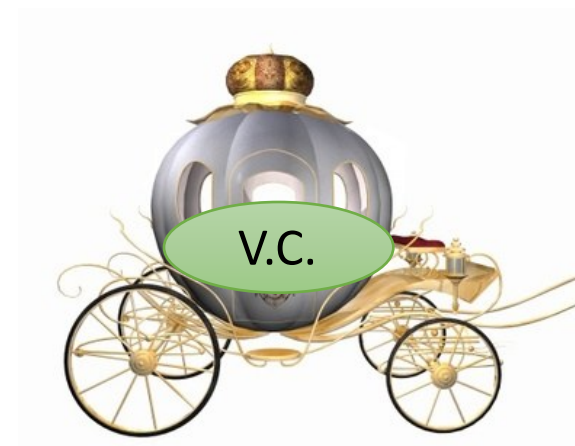


Cinderella: Turning Shabby X.509 Certificates into Elegant Anonymous Credentials with the Magic of Verifiable Computation

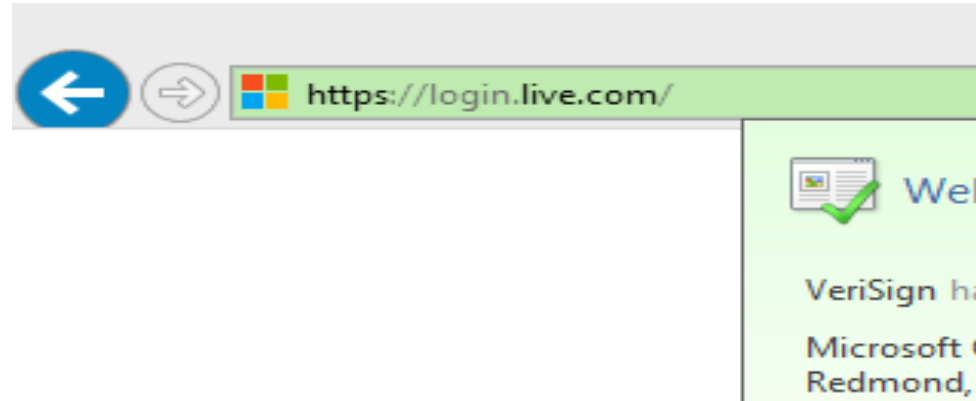


Antoine Delignat-Lavaud
Cédric Fournet, Markulf Kohlweiss,
Bryan Parno

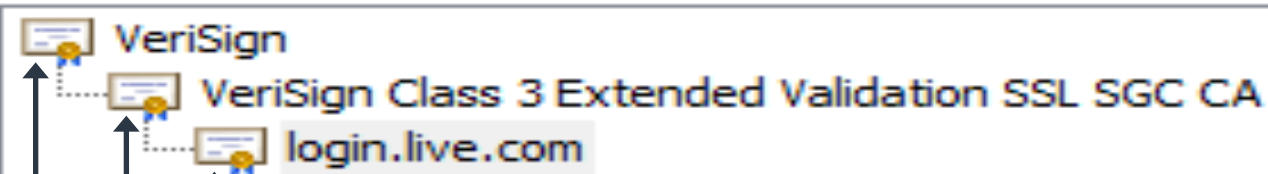


Microsoft®
Research

The X.509 Public Key Infrastructure (1988)



Certification path



Chain

Endpoint certificate

Intermediate Certificate Authority certificate

Root Certification Authority certificate

A screenshot of a "Certificate" dialog box with the "Details" tab selected. The "Show:" dropdown is set to "<All>". The dialog displays a list of fields and their corresponding values.

Field	Value
Version	V3
Serial number	61 5d aa d2 00 06 00 00 00 40
Signature algorithm	sha1RSA
Signature hash algorithm	sha1
Issuer	Microsoft Internet Authority
Valid from	15 May 2012 21:40:55
Valid to	15 May 2016 21:50:55
Subject	MSIT Machine Auth CA 2, red...
Public key	RSA (2048 Bits)
CA Version	V1.1
Previous CA Certificate Hash	23 b7 d0 ed 68 91 ef 66 9b e1...
Subject Key Identifier	eb db 11 5e f8 09 9e d8 d6 62...
Certificate Template Name	SubCA
Key Usage	Digital Signature, Certificate Si...
Authority Key Identifier	KeyID=2a 4d 97 95 5d 34 7e ...
CRL Distribution Points	[1]CRL Distribution Point: Distr...
Authority Information Access	[1]Authority Info Access: Acc...
Basic Constraints	Subject Type=CA, Path Leng...
Thumbprint algorithm	sha1
Thumbprint	ef 86 b4 13 f0 fc 25 ac 51 2b ...

X.509 Authentication

Certificate Authority

certificates + private keys

authorized root certificates (data)

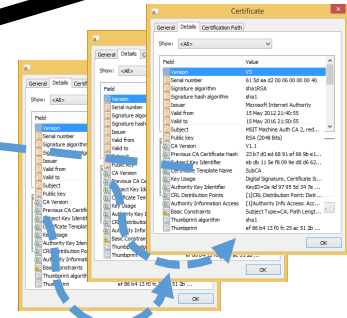
certificate validation program



Authentication challenge

Sign(challenge, private key)

(1-3KB / certificate)



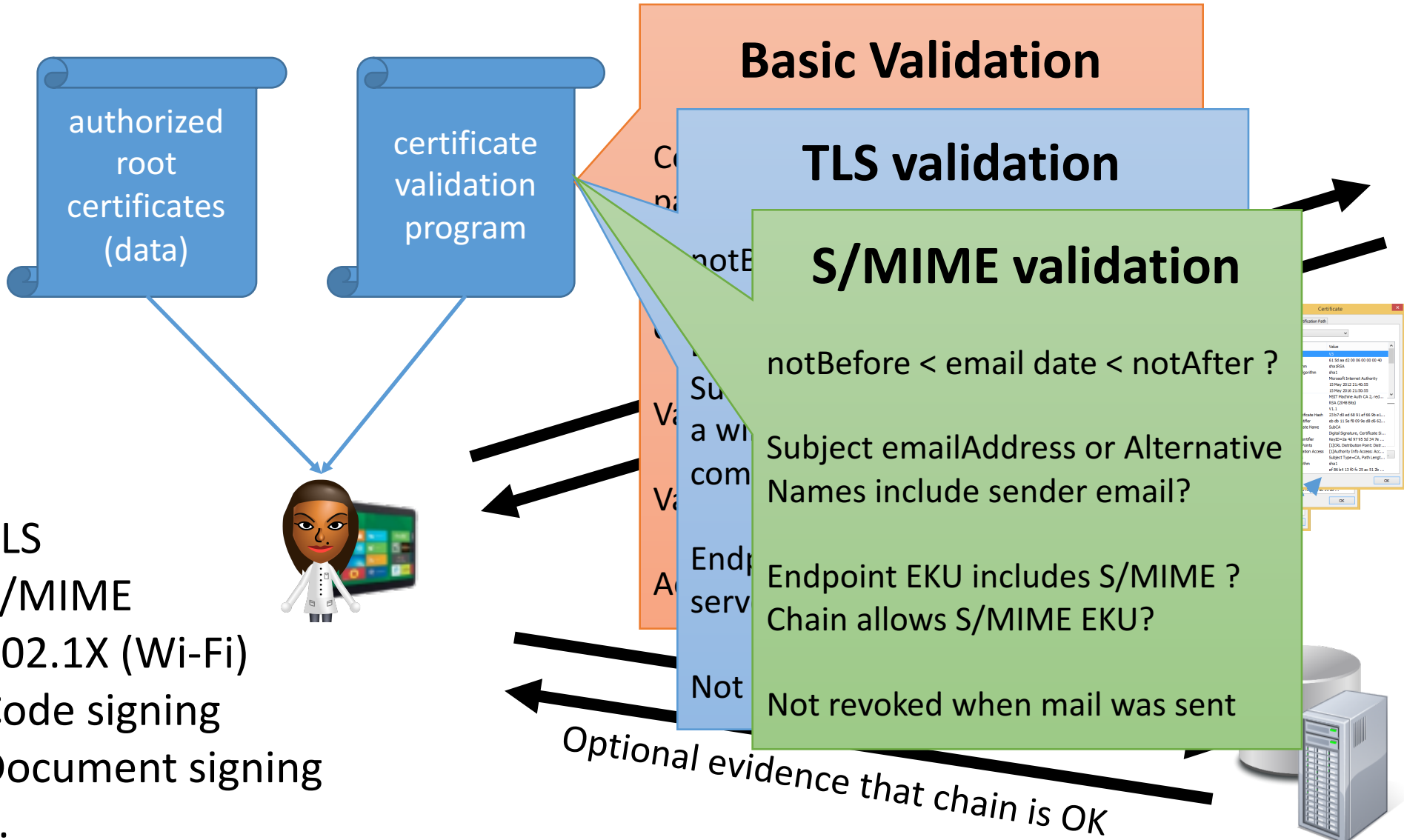
Optional evidence that chain is OK



OCSF, Certificate Transparency, Perspectives...

X.509 Problem: Application Heterogeneity

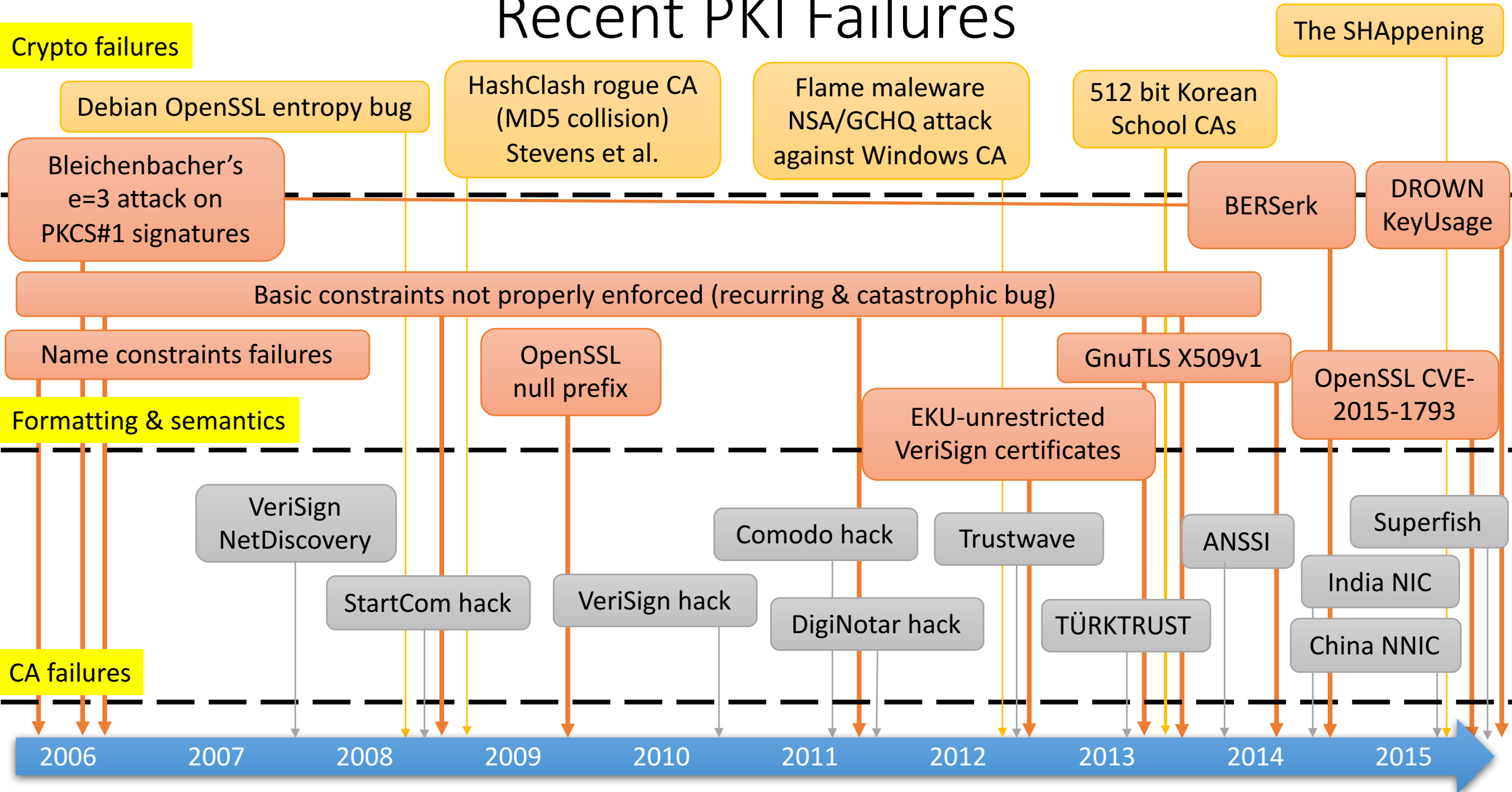
certificates + private keys



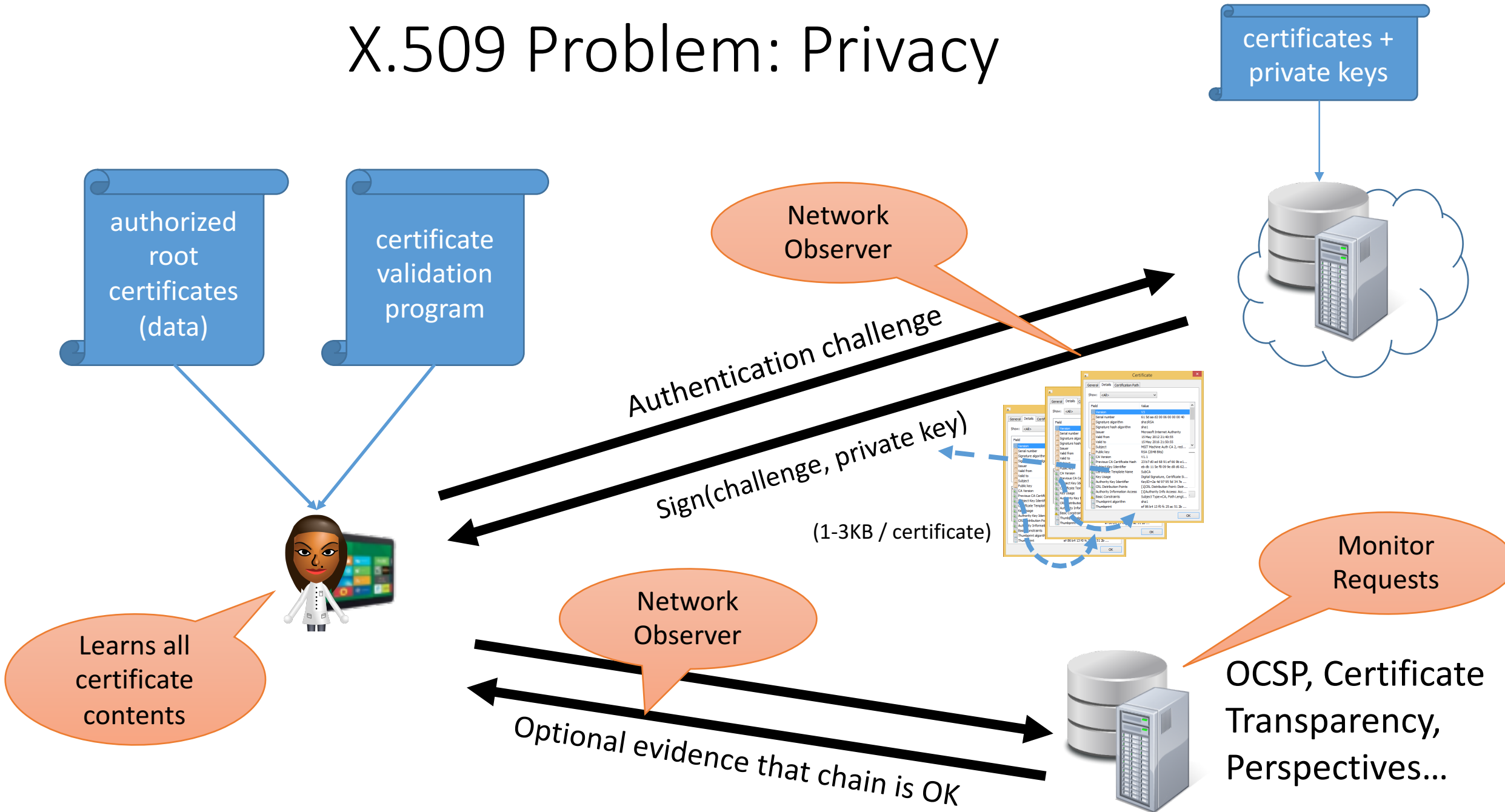
- TLS
- S/MIME
- 802.1X (Wi-Fi)
- Code signing
- Document signing
- ...

OCSP, Certificate Transparency, Perspectives...

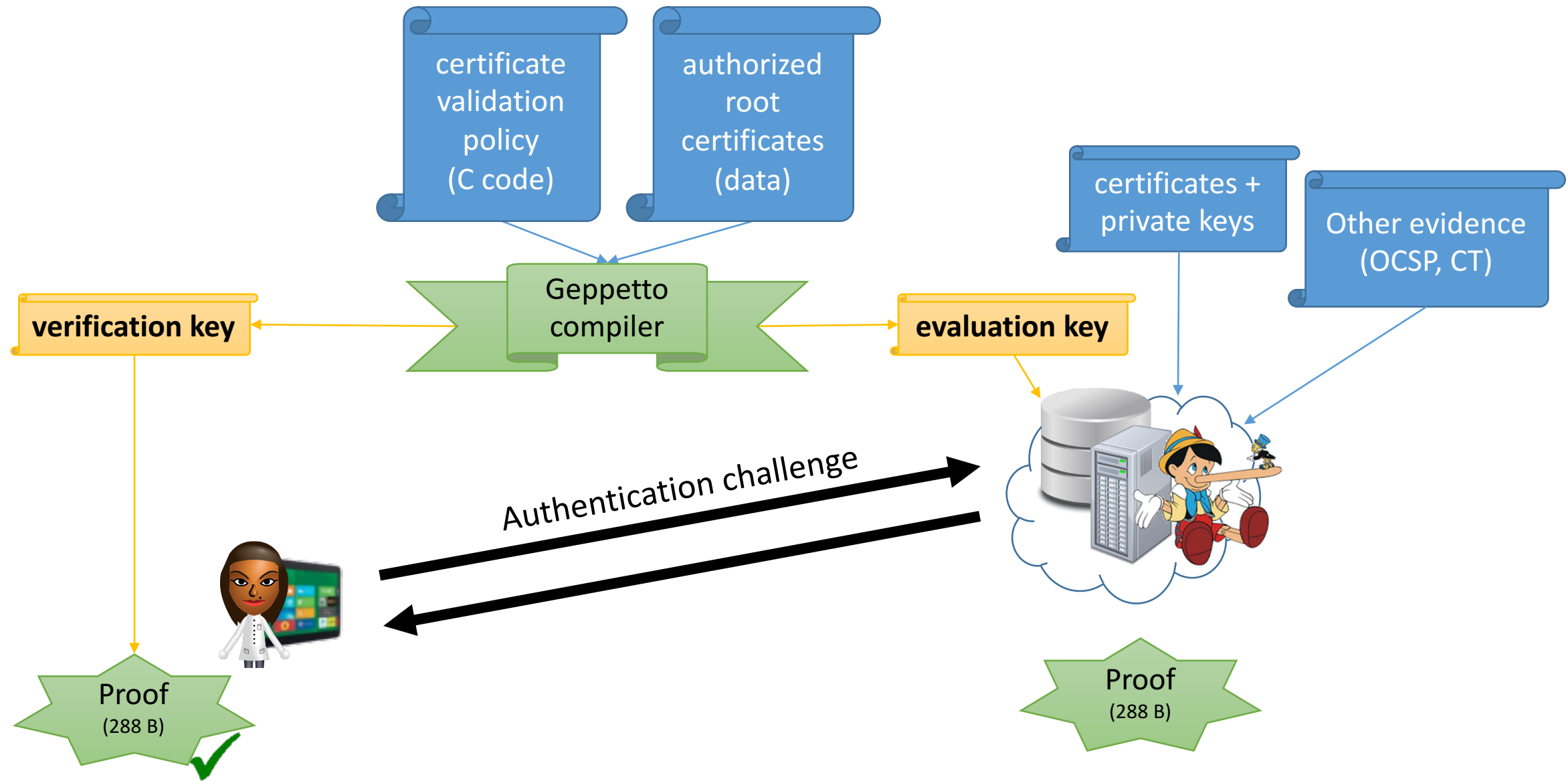
Recent PKI Failures



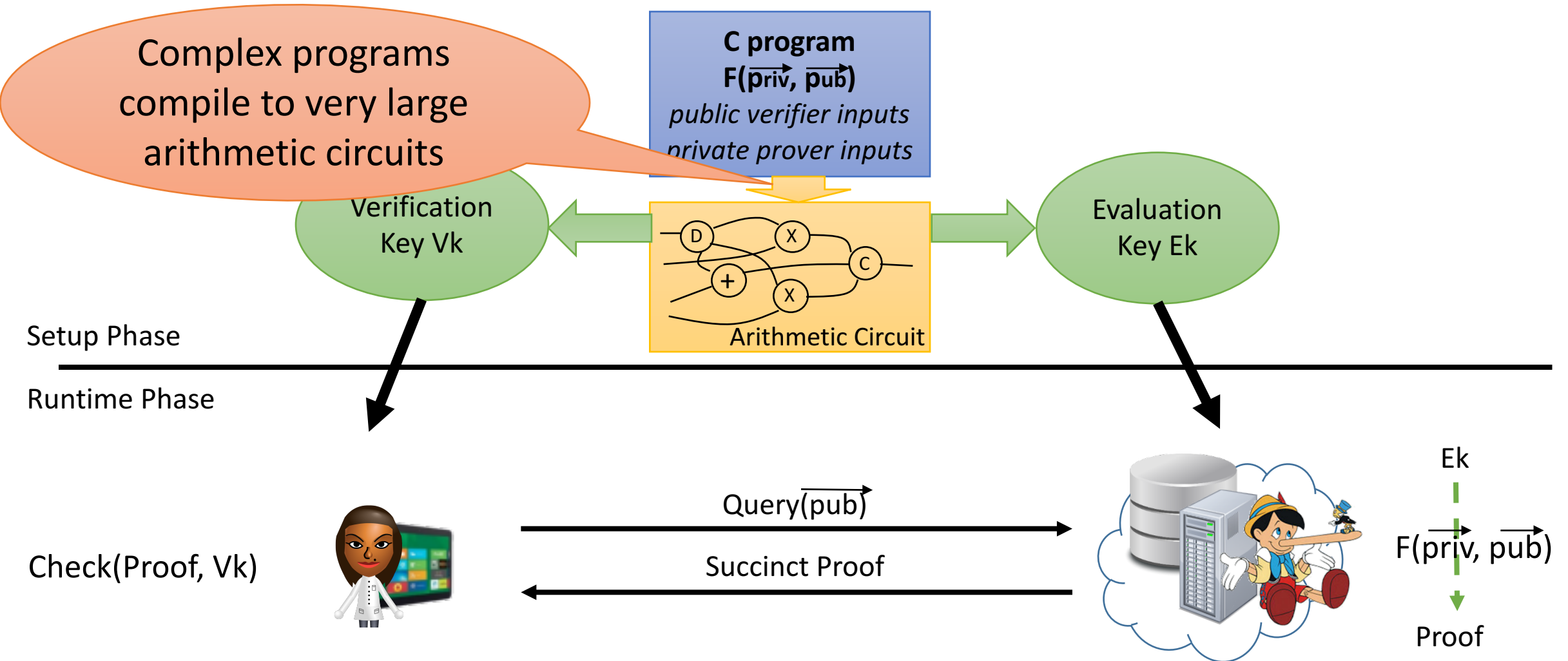
X.509 Problem: Privacy



Cinderella: Main Idea



Computation Outsourcing with Pinocchio



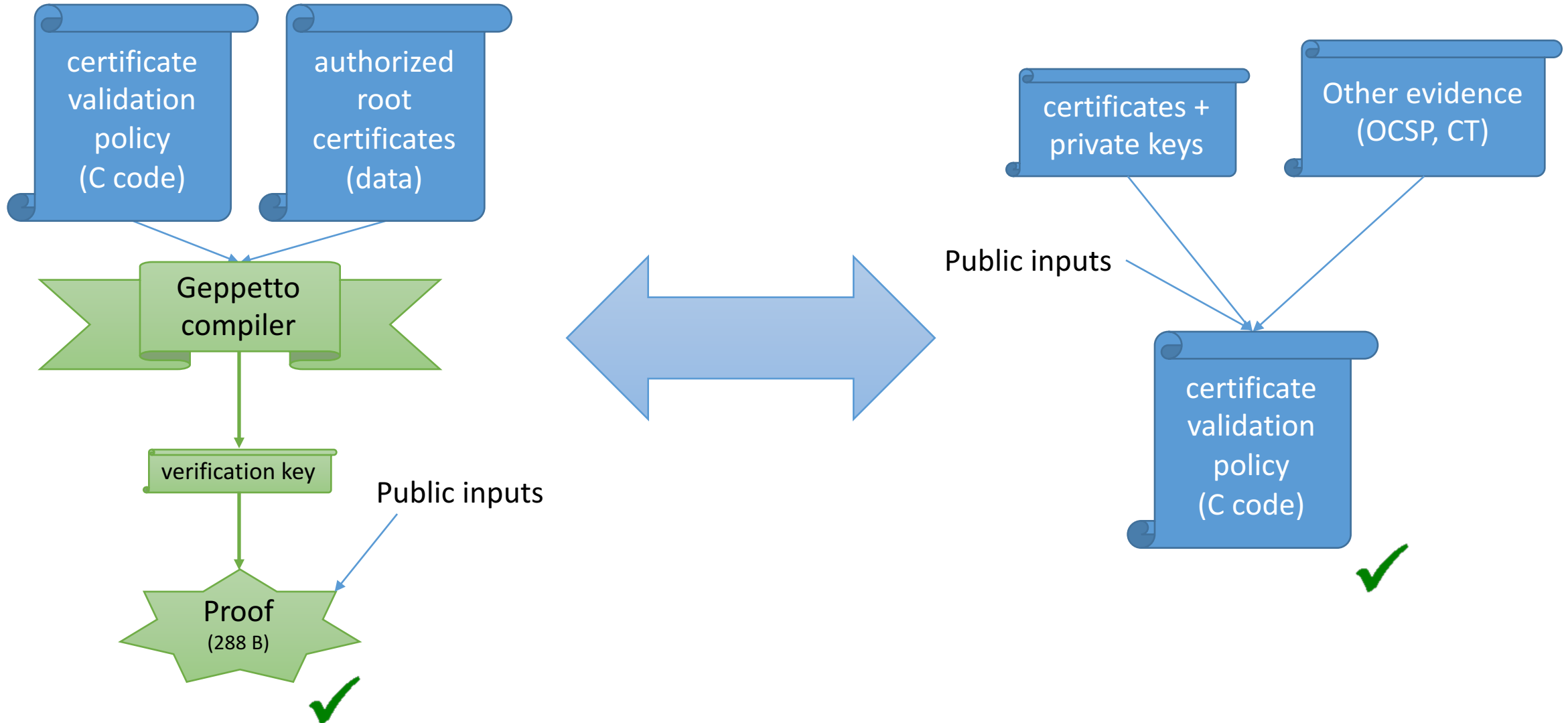
Cinderella: Contributions

- 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>; seq {  
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>;  
    };  
  };  
  [...]  
}
```

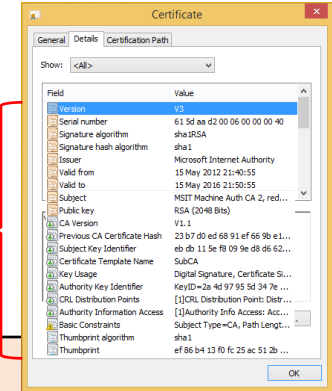
Constants

Variables

Variable lists

Template

Private inputs



Untrusted Native Parser
Parse certificate
Generate Prover Inputs

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

Template
Verifier
compiler

Produced Verifier (Fragment)

Variable list

```
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]);  
}
```

Constants

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

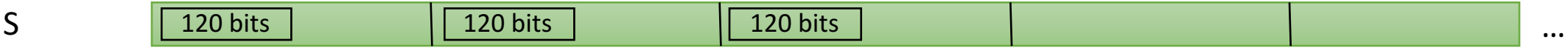
Verifying PKCS#1 RSA Signatures

$$S^e \bmod N = 1\text{ffffff}[...] \text{fffffkkkkk}[...] \text{kkkkkkXXXXXXXXXXXXXXXXXXXXXXXXXXXX}$$

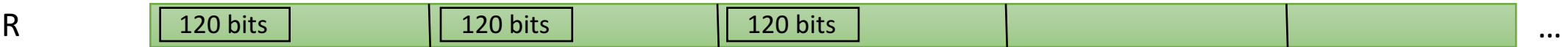
$$S^e = S \left((S^2)^2 \right) \dots$$

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

↑
Hash (computed before)

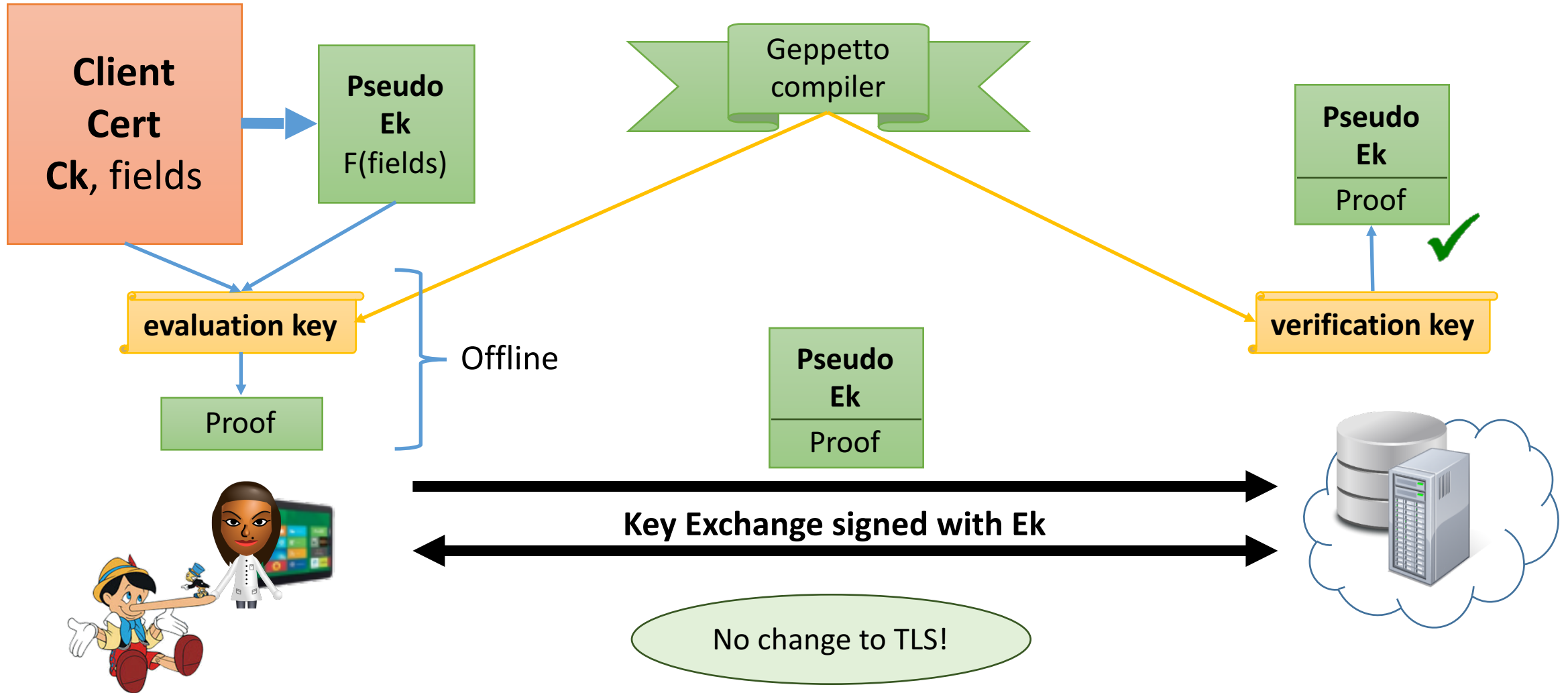


Private inputs Q and R -> $S^2 = Q \cdot N + R$ Verify the prover hints are valid

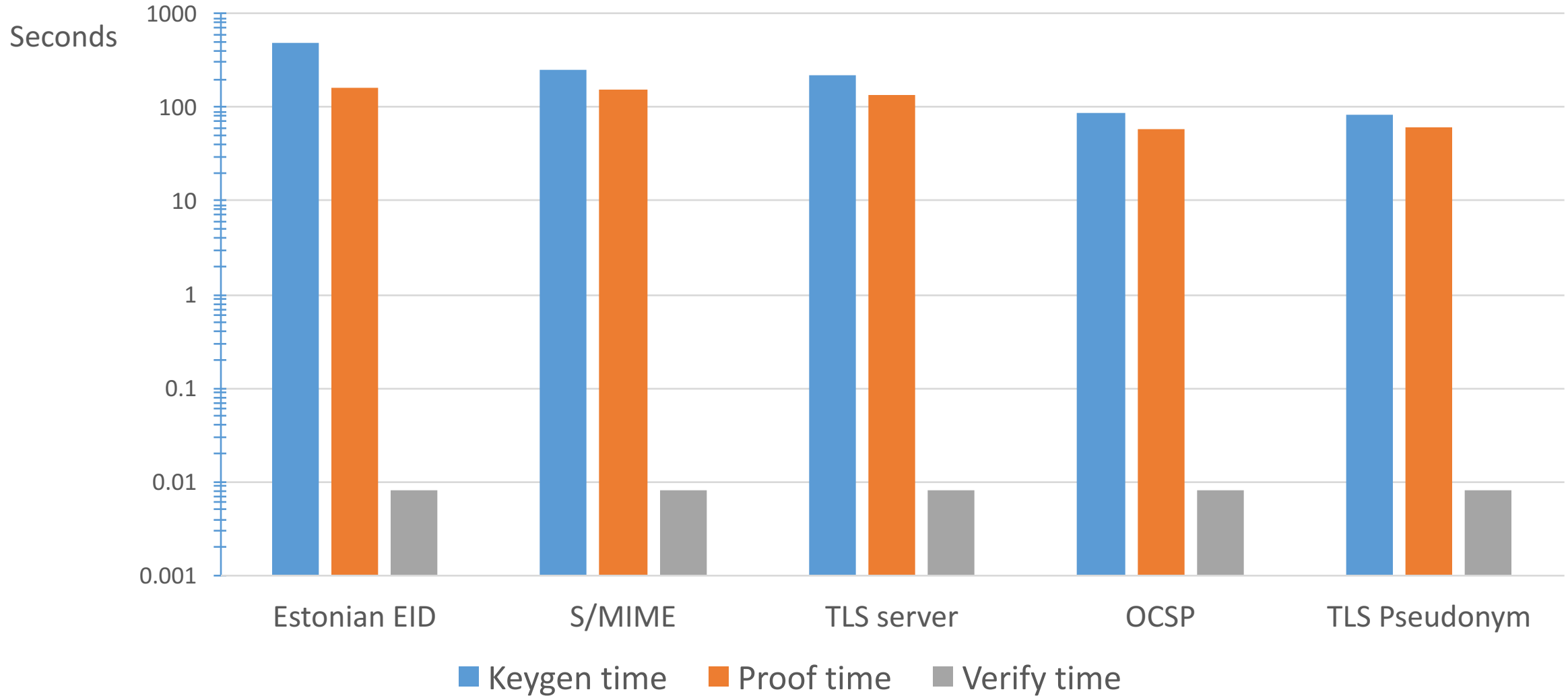


$$S \leftarrow R$$

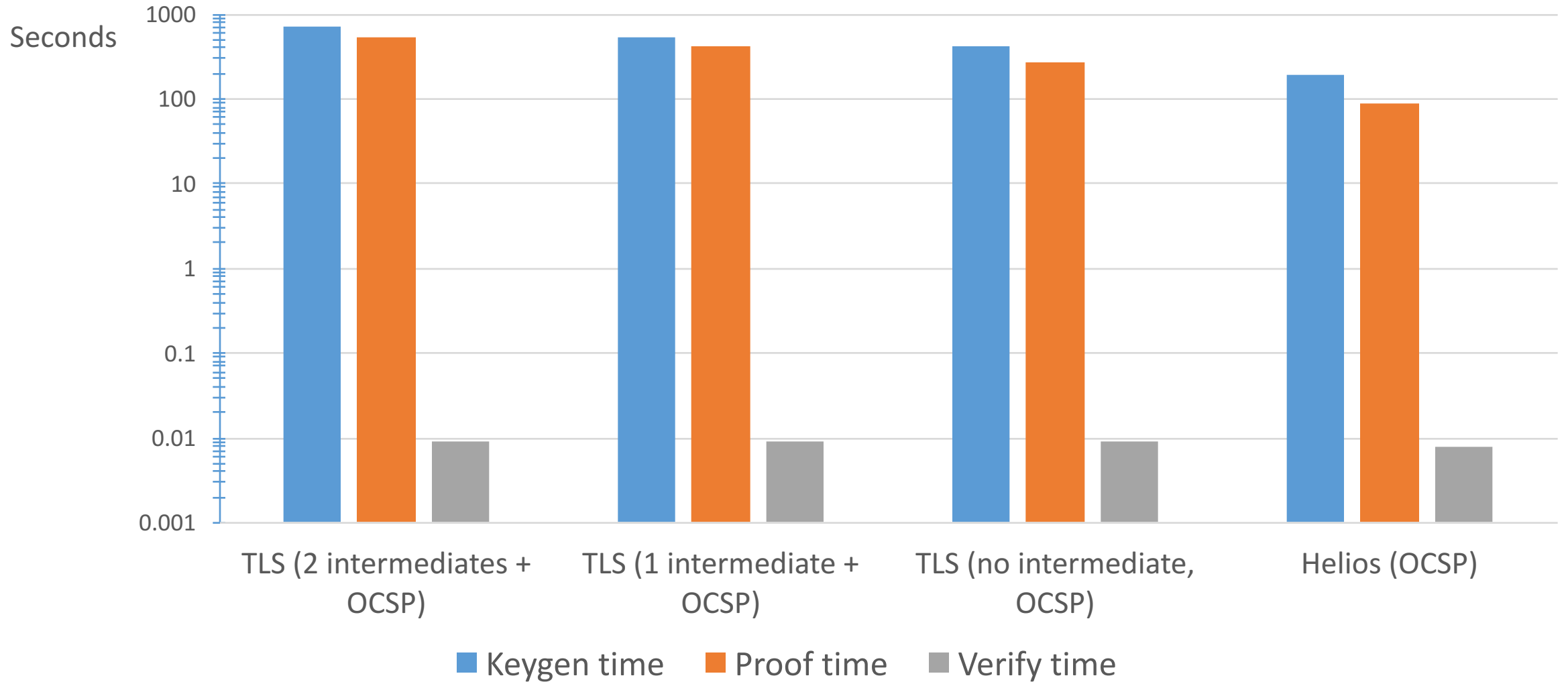
Application: TLS Client (with Offline Signing)



Single Template Evaluation (With Signature)



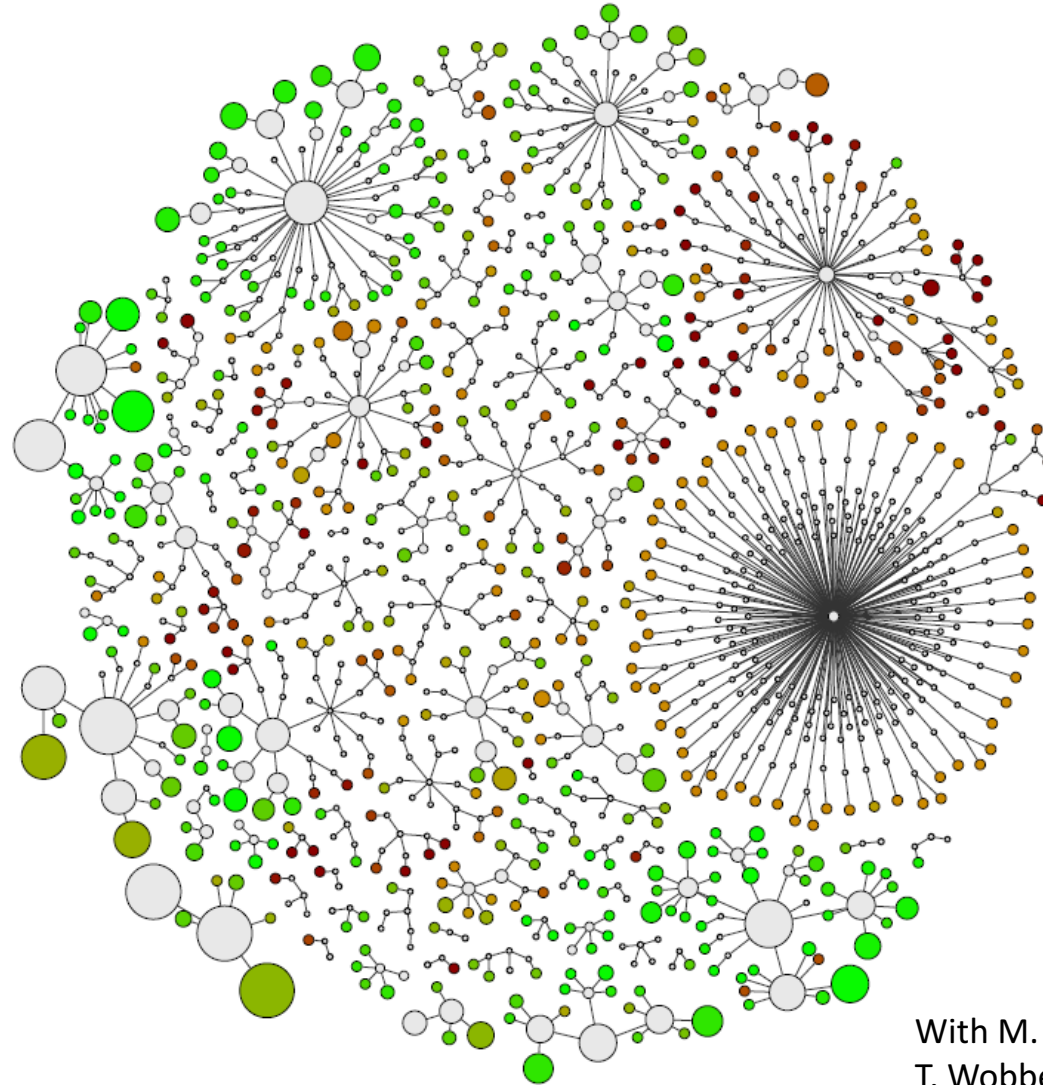
Application evaluation



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



$$\text{KeyGen}(R) \longrightarrow EK, VK$$

Generate the MultiQAP for R

Pick random s

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

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

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

$$\text{Verify}(VK_j, C_j)$$

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

and similarly for w and y

$$\text{Commit}(EK_j, u_j, o_j) \longrightarrow C_j$$

Generate the commitment:

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

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

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

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

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

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

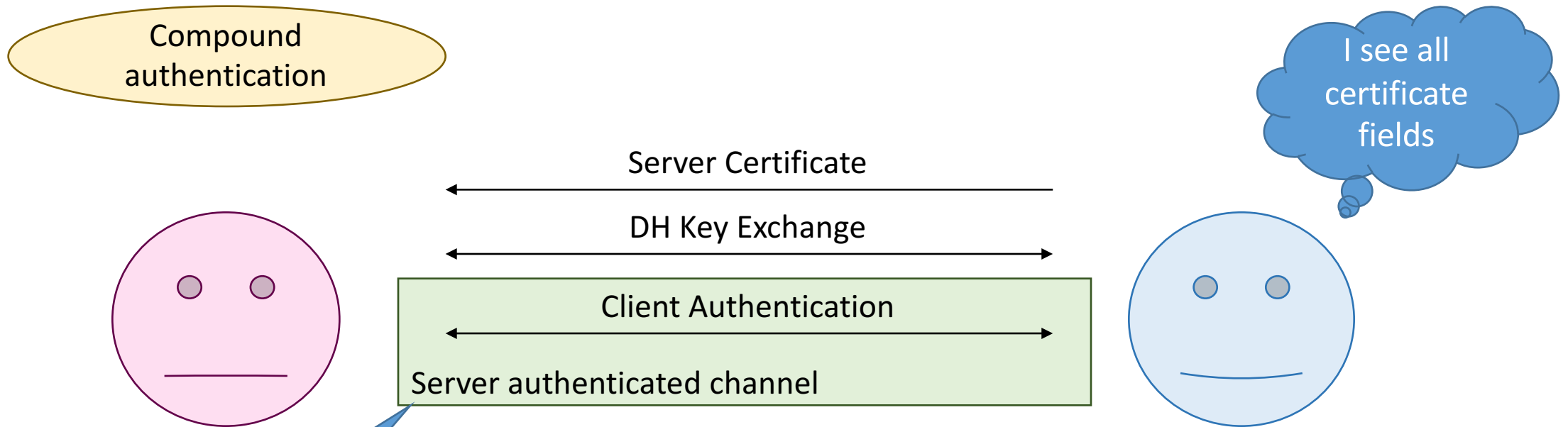
$$\text{Proof is } (g^{v(s)}, g^{w(s)}, g^{y(s)}, g^{h(s)})$$

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

$$\frac{e(g^{v(s)}, g^{w(s)})}{e(g^{y(s)}, g)} \stackrel{?}{=} e(g^{h(s)}, g^{d(s)}) \quad \left. \vphantom{\frac{e(g^{v(s)}, g^{w(s)})}{e(g^{y(s)}, g)}}} \right\} e(\cdot, \cdot) \text{ is a pairing:}$$

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

Workaround: Tunneling

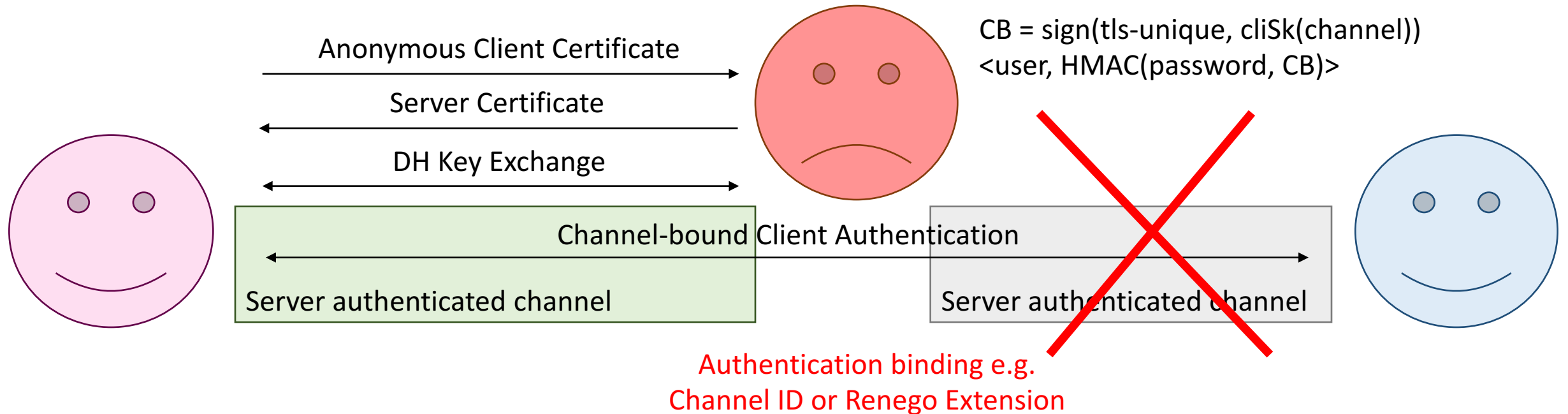


- TLS Renegotiation
- TLS 1.3 Handshake Encryption
- Server still sees all contents
- Not always possible (S/MIME, code and document signing)

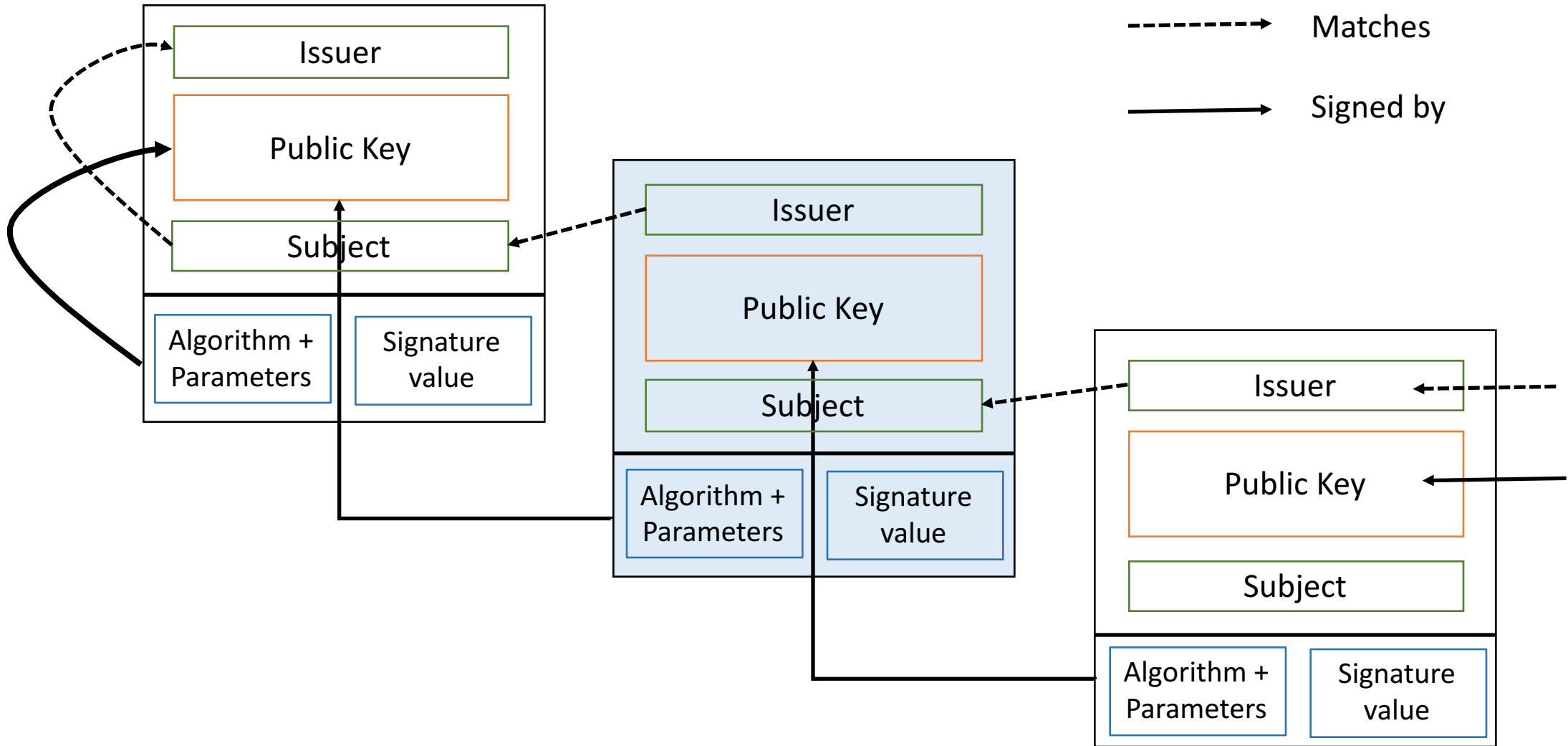
Usability and Privacy of PKI Authentication

Current Privacy Approach

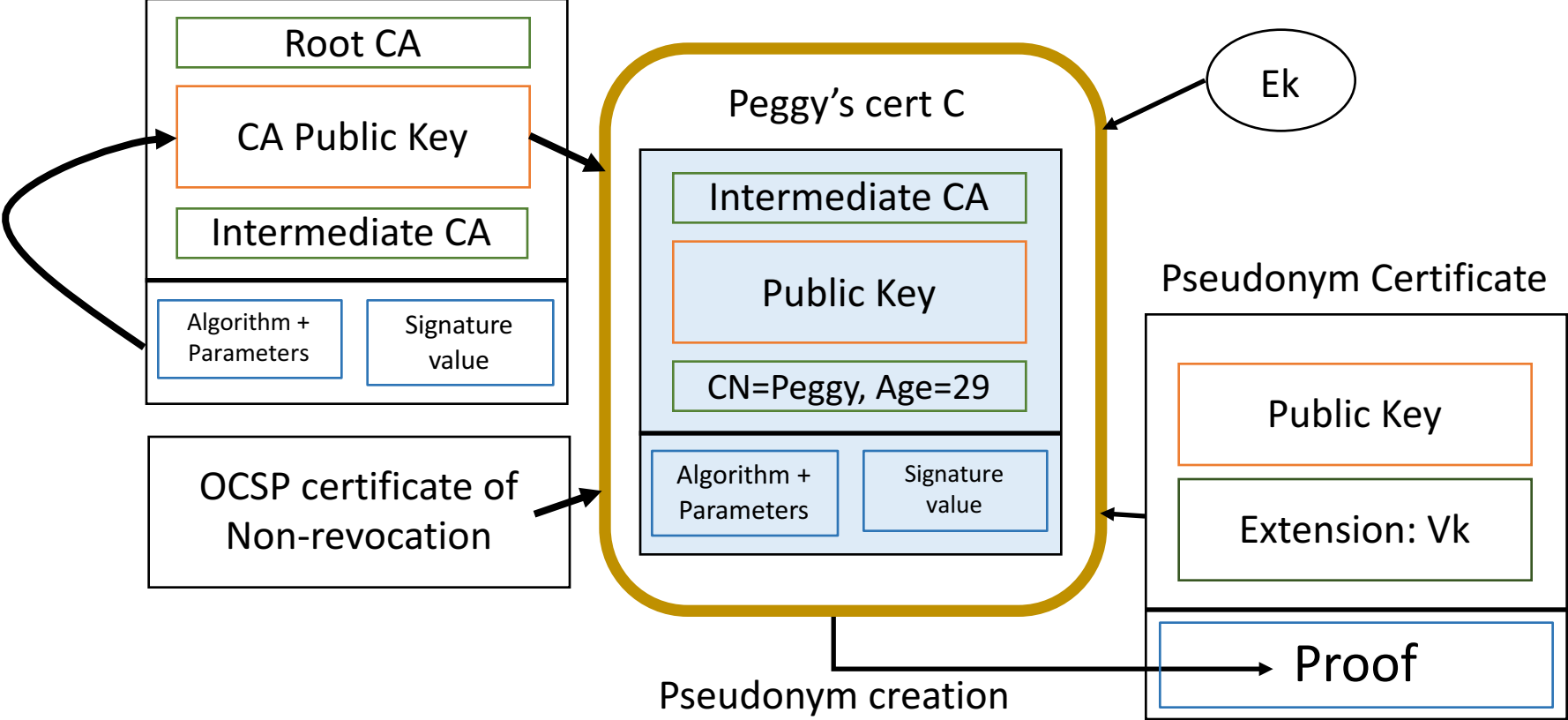
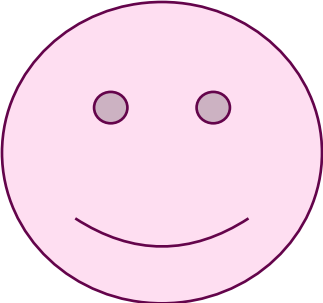
- User Unfriendly
- Complex
- Key Compromise Impersonation attacks



The Internet PKI



Deployment: X.509 Signature Scheme



ASN.1

Binary encoding standard

Ancient (1984)

<Tag, Length, Value>

Distinguished rules (DER):
unique serialization

```
SEQUENCE (3 elem)
  SEQUENCE (8 elem)
    [0] (1 elem)
      INTEGER 2
    INTEGER (141 bit) 1492258819486064224988303096848576164759414
    SEQUENCE (2 elem)
      OBJECT IDENTIFIER 1.2.840.113549.1.1.5
      NULL
    SEQUENCE (2 elem)
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.10
          PrintableString AlphaSSL
        SET (1 elem)
          SEQUENCE (2 elem)
            OBJECT IDENTIFIER 2.5.4.3
            PrintableString AlphaSSL CA - G2
    SEQUENCE (2 elem)
      UTCTime 2013-06-02 17:27:55 UTC
      UTCTime 2017-06-02 17:27:55 UTC
    SEQUENCE (2 elem)
      SET (1 elem)
        SEQUENCE (2 elem)
          Offset: 132
          Length: 2+31
          (constructed) (1 elem)
          Value:
            OBJECT IDENTIFIER 2.5.4.3
            PrintableString *.ht.vc
    SEQUENCE (2 elem)
      OBJECT IDENTIFIER 1.2.840.113549.1.1.1
      NULL
    BIT STRING (1 elem)
      SEQUENCE (2 elem)
        INTEGER (2048 bit) 25070016126400689348179857701190619
        INTEGER 65537
  [3] (1 elem)
    SEQUENCE (9 elem)
      SEQUENCE (3 elem)
        OBJECT IDENTIFIER 2.5.29.15
        BOOLEAN true
        OCTET STRING (1 elem)
          BIT STRING (3 bit) 101
      SEQUENCE (2 elem)
        OBJECT IDENTIFIER 2.5.29.32
        OCTET STRING (1 elem)
```

```
30 82 04 92 30 82 03 7A A0 03 02 01 02 02 12 11
21 5A C2 85 BD 0A 8C 58 07 4F 22 B4 89 04 29 87
76 30 0D 06 09 2A 86 48 86 F7 0D 01 01 05 05 00
30 2E 31 11 30 0F 06 03 55 04 0A 13 08 41 6C 70
68 61 53 53 4C 31 19 30 17 06 03 55 04 03 13 10
41 6C 70 68 61 53 53 4C 20 43 41 20 2D 20 47 32
30 1E 17 0D 31 33 30 36 30 32 31 37 32 37 35 35
5A 17 0D 31 37 30 36 30 32 31 37 32 37 35 35 5A
30 35 31 21 30 1F 06 03 55 04 0B 13 18 44 6F 6D
61 69 6E 20 43 6F 6E 74 72 6F 6C 20 56 61 6C 69
64 61 74 65 64 31 10 30 0E 06 03 55 04 03 14 07
2A 2E 68 74 2E 76 63 30 82 01 22 30 0D 06 09 2A
86 48 86 F7 0D 01 01 01 05 00 03 82 01 0F 00 30
82 01 0A 02 82 01 01 00 C6 97 C0 88 C6 30 A5 7A
0C 68 DA 22 F6 31 57 9C 9B 27 80 BB CD B9 D9 81
77 BF 6D 11 77 BE 9A 14 14 18 CB BB 38 C4 90 74
0D 17 73 2C DF 4E 34 F1 B4 C1 97 31 42 F5 DA 7E
ED B6 76 B6 D1 9D 78 4F D2 0F 31 27 AA 64 7E B7
DC 88 63 BF 9F 00 02 BD 68 98 29 A8 36 B1 68 2B
9D 05 AF A5 73 54 46 62 FE 7E A0 D4 D8 AD BF F5
1A CC 3F B7 22 E5 4B 52 F9 38 26 98 5D D6 07 F0
CB 0C 1E FF 43 E2 A9 AD BB B1 CA 83 A0 33 4F BA
76 4C 1E CA D9 A2 C4 86 F2 47 90 9B 98 92 57 76
F9 EE 5D 22 77 6F EB A3 EE 11 86 D2 13 C4 50 1C
90 09 62 D5 22 8E DF EB 51 8B F7 3E 66 B9 45 BC
76 13 45 CE 92 59 AD 27 1B B3 E3 25 1D 0A 13 FD
CA 94 BF 7D 60 37 03 00 20 87 D8 75 B2 49 03 5A
CF 96 17 79 C6 7C 46 6E D1 C4 67 D9 E1 C9 64 7B
8A 72 0C 3A 2A 6E C6 E4 45 6F AD A9 D7 28 ED 54
B3 F9 58 DB 21 B3 4D D1 02 03 01 00 01 A3 82 01
A1 30 82 01 9D 30 0E 06 03 55 1D 0F 01 01 FF 04
04 03 02 05 A0 30 49 06 03 55 1D 20 04 42 30 40
30 3E 06 06 67 81 0C 01 02 01 30 34 30 32 06 08
2B 06 01 05 05 07 02 01 16 26 68 74 74 70 73 3A
2F 2F 77 77 77 2E 67 6C 6F 62 61 6C 73 69 67 6E
2E 63 6F 6D 2F 72 65 70 6F 73 69 74 6F 72 79 2F
30 19 06 03 55 1D 11 04 12 30 10 82 07 2A 2E 68
74 2E 76 63 82 05 68 74 2E 76 63 30 09 06 03 55
```

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], comp1);

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

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