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
The X.509 Public Key Infrastructure (1988)

Chain

Endpoint certificate

Intermediate Certificate Authority certificate

Root Certification Authority certificate
X.509 Authentication

Certificate Authority

authorized root certificates (data)
certificate validation program

certificates + private keys

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
- …

Basic Validation

Correct OCTHEM encoding (injective parsing)
Valid basic constraints
Valid key usages
Acceptable algorithms and key sizes

TLS validation

notBefore < now() < notAfter
Domain == Subject CN?
Domain in Subject Alternative Names?
Matches a wildcard name?
Domain compatible with Name Constraints?
Endpoint EKU includes TLS client/server?
Chain allows TLS EKU?
Not revoked now

S/MIME validation

notBefore < email date < notAfter
Subject emailAddress or Alternative Names include sender email?
Endpoint EKU includes S/MIME?
Chain allows S/MIME EKU?
Not revoked when mail was sent

Optional evidence that chain is OK

authorized root certificates (data)
certificate validation program
certificates + private keys

OCSP, Certificate Transparency, Perspectives…
Recent PKI Failures

Crypto failures

- Debian OpenSSL entropy bug
- Bleichenbacher’s e=3 attack on PKCS#1 signatures
- HashClash rogue CA (MD5 collision) Stevens et al.
- Flame maleware NSA/GCHQ attack against Windows CA
- 512 bit Korean School CAs

Name constraints failures

- Basic constraints not properly enforced (recurring & catastrophic bug)

Formatting & semantics

- OpenSSL null prefix

CA failures

- VeriSign NetDiscovery
- StartCom hack
- VeriSign hack
- Comodo hack
- DigiNotar hack
- Trustwave
- ANSSI
- Superfish
- India NIC
- China NNIC

OpenSSL

- OpenSSL CVE-2015-1793
- GnuTLS X509v1

The SHAppening

- BERSerk
- DROWN KeyUsage
X.509 Problem: Privacy

authorized root certificates (data)
certificate validation program

Network Observer
Authentication challenge
Sign(challenge, private key)
(1-3KB / certificate)

Network Observer

Learns all certificate contents

Monitor Requests
OCSP, Certificate Transparency, Perspectives...

certificates + private keys

Network Observer
Optional evidence that chain is OK
Cinderella: Main Idea

- Verification key
- Geppetto compiler
- Evaluation key

- Certificate validation policy (C code)
- Authorized root certificates (data)
- Certificates + private keys
- Other evidence (OCSP, CT)

Authentication challenge
Computation Outsourcing with Pinocchio

Complex programs compile to very large arithmetic circuits

Setup Phase

C program
\[ F(\text{priv}, \text{pub}) \]

public verifier inputs
private prover inputs

Runtime Phase

Verification Key \( V_k \)

Evaluation Key \( E_k \)

Succinct Proof

Check(Proof, \( V_k \))

[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

- Certificate validation policy (C code)
- Authorized root certificates (data)
- Other evidence (OCSP, CT)
- Certificate validation policy (C code)
- Public inputs
- Public inputs
- Geppetto compiler
- Verification key
- Proof (288 B)
Compiling Certificate Templates

seq { seq {
  # Version
  tag<0>: const<2L>;
  # Serial Number
  var<int, serial, 10, 20>;
  # Signature Algorithm
  seq {
    const<01.2.840.113549.1.1.5>;
    const>null; }
  }
  # Issuer
  seq { set { seq {
    const<02.5.4.10>;
    const<printable:"AlphaSSL">; }}; set { seq { const<02.5.4.3>;
    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>;
      }; 
    };
};

[…]

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

Private inputs
Produced Verifier (Fragment)

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

**Append(byte)**
Add given byte to the hashing buffer

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

**Compression**

- Output = hash of ASN.1 formatted certificate contents
Verifying PKCS#1 RSA Signatures

\[ S^e \mod N = 1\text{fffffff}\ldots\text{kkkkkk}\ldots \text{XXXXXXXXXXXXXXXXXXXXX} \]

\[ S^e = S ((S^2)^2) \ldots \]

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

Private inputs \( Q \) and \( R \) ->

\[ S^2 = Q*N + R \]

Verify the prover hints are valid

\[ S <- R \]
Application: TLS Client (with Offline Signing)

Client Cert Ck, fields

Pseudo Ek F(fields)

Geppetto compiler

Pseudo Ek Proof

Proof

evaluation key

Offline

verification key

Key Exchange signed with Ek

No change to TLS!
Application evaluation

Seconds

- TLS (2 intermediates + OCSP)
- TLS (1 intermediate + OCSP)
- TLS (no intermediate, OCSP)
- Helios (OCSP)

Keygen time  Proof time  Verify time
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($R$) $\rightarrow$ EK, VK
Generate the MultiQAP for $R$
Pick random $s$
Compute $EK = \{EK_j\}, \{g^s\}$

$EK_j = \left\{ g^{v_k(s)}, g^{w_k(s)}, g^{y_k(s)} \middle| \begin{array}{l}
g^{\alpha_{j,v}v_k(s)}, g^{\alpha_{j,w}w_k(s)}, g^{\alpha_{j,y}y_k(s)} \end{array} \right\}_{k \in J}$
Compute $VK = (g^{d(s)} = g^{\prod_i(s-i)})$

Commit($EK_j, u_j, o_j$) $\rightarrow$ $C_j$
Generate the commitment:
$v^j(s) = \sum_{k \in J} u_k v_k(s) + o_j w(s), \text{ similarly for } w \text{ and } y$
$C_j = (g^{v^j(s)}, g^{y^j(s)}, g^{jv^j(s)}, g^{yw^j(s)}, g^{ajwjv^j(s)}, g^{ajyw^j(s)}, g^{ajyw^j(s)})$
$v(s) = \prod v^j(s) \text{ and similarly for } w \text{ and } y$

Prove($EK, u, o$) $\rightarrow$ $\pi$
Find $h(x)$ s.t. $h(x) \ast d(x) = v(x) \ast w(x) - y(x)$
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($VK_j, C_j$)
$e(g^{v^j(s)}, g^{\alpha_{j,v}w(s)}) = e(g^{\alpha_{j,v}v^j(s)}, g)$
and similarly for $w$ and $y$

Verify($VK, C, \pi$) $\rightarrow$ {Yes, No}
$\frac{e(g^{v(s)}, g^{w(s)})}{e(g^{y(s)}, g)} = e(g^{h(s)}, g^{d(s)})$
$e(\cdot, \cdot)$ is a pairing:
$e(g^p, g^q) = e(g, g)^{pq}$
Workaround: Tunneling

- Compound authentication
- Server Certificate
- DH Key Exchange
- Client Authentication
- Server authenticated channel

- Performance overhead of tunneling
- I see all certificate fields

- 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

- User Unfriendly
- Complex
- Key Compromise Impersonation attacks

Anonymous Client Certificate

Server Certificate

DH Key Exchange

Channel-bound Client Authentication

Server authenticated channel

Authentication binding e.g.
Channel ID or Renego Extension

Current Privacy Approach

CB = sign(tls-unique, cliSk(channel))
$user, \text{HMAC(password, CB)}$
The Internet PKI
Deployment: X.509 Signature Scheme

- Root CA
- CA Public Key
- Intermediate CA
- Algorithm + Parameters
- Signature value
- OCSP certificate of Non-revocation

Peggy's cert C
- Intermediate CA
- Public Key
- Algorithm + Parameters
- Signature value
- CN=Peggy, Age=29

Ek
- Pseudonym Certificate
- Public Key
- Extension: Vk
- Proof
- Pseudonym creation
**ASN.1**

Binary encoding standard

Ancient (1984)

<Tag, Length, Value>

Distinguished rules (DER):
- unique serialization

```
SEQUENCE (3 clm)
  [0] (1 clm)
    INTEGER 2
      INTEGER (141 bit) 1492255819486064224983303096858476164759414
  [1] (2 clm)
    OBJECT IDENTIFIER 1.2.540.113549.1.1.5
      NULL
  [2] (2 clm)
    SEQUENCE (2 clm)
      OBJECT IDENTIFIER 2.5.6.3
        PrintableString AlphaSSL
    SET (1 clm)
      SEQUENCE (2 clm)
        OBJECT IDENTIFIER 2.5.4.10
          PrintableString AlphaSSL
        SEQUENCE (2 clm)
          OBJECT IDENTIFIER 2.5.6.3
            PrintableString AlphaSSL CA - G2
      SEQUENCE (2 clm)
        UTCTime 2013-06-02 17:27:55 UTC
        UTCTime 2017-06-02 17:27:55 UTC
    SET (1 clm)
      SEQUENCE (2 clm)
        Offset: 182
        Length: 2431
        (constructed) (len)
        Value: TIER 2.5.4.3
          0 * .h.t.v.c
      SEQUENCE (2 clm)
        OBJECT IDENTIFIER 1.2.540.113549.1.1.1
        NULL
      BIT STRING (1 clm)
        OBJECT IDENTIFIER 1.2.540.113549.1.1.1
        INTEGER 250700161264000839348179857701196019
        INTEGER 65537
      [3] (1 clm)
        SEQUENCE (9 clm)
          OBJECT IDENTIFIER 2.5.29.15
          BOOLEAN true
          OCTET STRING (1 clm)
            101
      SEQUENCE (2 clm)
        OBJECT IDENTIFIER 2.5.29.32
        OCTET STRING (1 clm)
```
Checking RSA Signatures

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

```c
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]);
}