Making Verifiable Computation Useful

Bryan Parno

Carnegie Mellon University
Rapid Perf Improvements

100x100 matrix mult.

Verifier Latency

Prover Overhead

Cost fell 23 orders of magnitude in 6 years

<10 ms!
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

X.509

VC
The X.509 Public Key Infrastructure (1988)

Chain

Endpoint certificate

Intermediate Certificate Authority certificate

Root Certification Authority certificate
X.509 Authentication

Authorized root certificates (data)

Certificate validation program

Certificate Authority

certificates + private keys

Authentication challenge

Sign(challenge, private key)

(1-3 KB/certificate)

Optional evidence that chain is OK

OCSP, Certificate Transparency
X.509 Problem: App Heterogeneity

- Authorized root certificates (data)
- Certificate validation program
- Certificates + private keys

Basic Validation

TLS Validation

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

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

OCSP, Certificate Transparency

Optional evidence that chain is OK
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 malware NSA/GCHQ attack against Windows CA
- 512 bit Korean School CAs
- HashClash rogue CA
- Stevens et al.
- Flame malware
- NSA/GCHQ attack against Windows CA

Formatting & semantics
- Name constraints failures
- Basic constraints not properly enforced (recurring & catastrophic bug)
- OpenSSL null prefix
- EKU-unrestricted VeriSign certificates
- GnuTLS X509v1
- OpenSSL CVE-2015-1793
- Superfish
- India NIC
- China NNIC

CA failures
- VeriSign NetDiscovery
- StartCom hack
- VeriSign hack
- DigiNotar hack
- TÜRKTRUST
- Comodo hack
- Trustwave
- ANSSI
- India NIC
- China NNIC
- The SHAppening

X.509 Problem: Privacy violations

Many anonymous credential systems solve this, but ~0 are used today
Cinderella: Main Idea

Geppetto compiler [IEEE S&P ‘15]

Verification key
- Authorized root certificates (data)
- Certificate validation program

Evaluation key
- Certificates + private keys
- Other evidence (e.g. OCSP)

Authentication challenge

Proof (288 B)
Computation Outsourcing with Pinocchio

Setup Phase

- Complex programs compile to large arithmetic circuits

Runtime Phase

- Verification key (VK)
- Evaluation key (EK)
- Query(pub)
- Succinct Proof
- Evaluate(F(priv, pub), EK)

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

```plaintext
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
- `tag<0>`
- `const<2L>`
- `var<int, serial, 10, 20>`
- `const<O1.2.840.113549.1.1.5>`
- `const<null>`
- `const<O2.5.4.10>`
- `const<printable:"AlphaSSL">`
- `const<O2.5.4.3>`
- `const<printable:"AlphaSSL CA-G2">`

### Variable lists
- `varlist<subject, 2, 4>`
- `set { seq { var<oid, subjectoid, 3, 10>;
    var<x500, subjectval, 2, 31>;
} }`;

### Constants
- `tag<0>`
- `const<2L>`
- `var<int, serial, 10, 20>`
- `const<O1.2.840.113549.1.1.5>`
- `const<null>`
- `const<O2.5.4.10>`
- `const<printable:"AlphaSSL">`
- `const<O2.5.4.3>`
- `const<printable:"AlphaSSL CA-G2">`

---

### Untrusted Native Parser
- Parse certificate
- Generate Prover Inputs

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

---

Private inputs
Verifying PKCS#1 RSA Signatures

\[ S^e \mod N = 1fffffffff[...]ffffffffkkkk[...]kkkkkyyyyyyyyyyyyyyyyyyyyy \]

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

### Private inputs Q and R

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

Verify prover hints are valid

<table>
<thead>
<tr>
<th>S</th>
<th>120 bits</th>
<th>120 bits</th>
<th>120 bits</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>S^2</td>
<td>240+ bits</td>
<td>240+ bits</td>
<td>240+ bits</td>
<td>...</td>
</tr>
<tr>
<td>Q^N</td>
<td>240+ bits</td>
<td>240+ bits</td>
<td>240+ bits</td>
<td>...</td>
</tr>
<tr>
<td>R</td>
<td>120 bits</td>
<td>120 bits</td>
<td>120 bits</td>
<td>...</td>
</tr>
</tbody>
</table>

\[ S \leftarrow R \]
Application: TLS Client Authentication

Client Cert fields → Ephem Key F(fields) → Geppetto compiler [IEEE S&P '15] → Ephem Key F(fields) → Verification key

Ephem Key F(fields) → Proof

Evaluation key

Proof

Offline

Key Exchange signed with Ephem Key
Application evaluation

Seconds

<table>
<thead>
<tr>
<th>Description</th>
<th>Keygen time</th>
<th>Proof time</th>
<th>Verify time</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS (2 intermediates + OCSP)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS (1 intermediate + OCSP)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS (no intermediate, OCSP)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Helios (OCSP)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

+ 

Fully verified software

Secure verifiable computation

Trusted Platform Module (TPM)

Software Guard Extensions (SGX)

Ironclad Apps

- ~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