SCAPI: The Secure Computation API

Yehuda Lindell
Bar-Ilan University, Israel

TCC 2014 Rump Session
February 25, 2014
Most implementation projects are aimed at solving a **specific problem** more efficiently or with better security

SCAPI is an implementation project with no specific problem in mind (it is a **general-purpose** secure computation library)

SCAPI is **open source**; we have a long-term commitment (as long as we have money) to the project (bug fixes, additional functionality, improve existing implementations etc.)
SCAPI is written in Java
- Suitable for large projects, and quick implementation
- Portability (e.g., secure computation between a mobile device and a server)
- Existing libraries (e.g., Bouncy Castle)
- The JNI framework: can use libraries and primitives written in native code (and thus inherit their efficiency)
Design Principles

- **Flexibility:**
  - Cryptographers write protocols in abstract terms (OT, commitment, PRF, etc.)
  - SCAPI encourages implementation at this abstract level (work with any “DLOG group” and afterwards instantiate with concrete group and concrete library; e.g. EC-group from Miracl)
  - Can work at many different levels of abstraction, as desired

- **Extendibility:** can add support for any new libraries and implementation by providing wrappers that implement the defined interfaces (we are now adding openSSL)

- **Efficiency:** via JNI can access fast low-level libraries like Miracl, but work at the level of Java and with abstract objects

- **Ease of use:** SCAPI uses terminology that cryptographers are used to; SCAPI is well documented and has been written explicitly with other users in mind
Consider an oblivious transfer protocol that uses a group, a commitment scheme, and a hash function.

The theorem stating security of the protocol would say:

- Assume that DDH is hard in the group, the commitment is perfectly binding, and the hash function is collision resistant.
- Then, the OT protocol is secure.

SCAPI differentiates between security levels by defining hierarchies of interfaces, and protocol constructors can check them.
SCAPI defines **hierarchies of interfaces** for security levels.

```
  «interface»  
  HashSecLevel
  
  «interface»  
  DlogSecLevel
  
  «interface»  
  TargetCollisionResistant
  
  «interface»  
  CollisionResistant
  
  «interface»  
  CDH
  
  «interface»  
  DDH
  
  «interface»  
  Indistinguishable
  
  «interface»  
  NonMalleable
  
  «interface»  
  Cpa
  
  «interface»  
  Cca1
  
  «interface»  
  Cca2
  
  «interface»  
  EncSecLevel
  
  «interface»  
  Eav
  
  «interface»  
  Eav
```
The OT protocol receives a dlog group, commitment and hash function in its constructor.

It checks that:

- The dlog group is an instance of DDH
- The commitment is an instance of PerfectBinding
- The hash function is an instance of CollisionResistant

Security levels are also defined for protocols (semi-honest, covert, malicious, stand-alone, UC secure, and so on).
SCAPI has three layers

- Basic primitives (discrete log groups, PRFs, PRPs, hash, universal hash, etc.)
- Non-interactive schemes (symmetric and asymmetric encryption, MACs, signatures)
- Interactive protocols (oblivious transfer, garbled circuits, sigma protocols, ZK, ZKPOK, commitments, etc.)
  - We are continually adding: OT extensions for semi-honest (ACM CCS 13), JustGarble, wrapper for OpenSSL
public interface CramerShoupDDHEnc extends AsymmetricEnc, Cca2 {
}

public CramerShoupAbs(DlogGroup dlogGroup, CryptographicHash hash, SecureRandom random){
//The Cramer-Shoup encryption scheme must work with a Dlog Group that has DDH security level
//and a Hash function that has CollisionResistant security level. If any of this conditions is not
//met then cannot construct an object of type Cramer-Shoup encryption scheme; therefore throw exception.
if(!(dlogGroup instanceof DDH)){
    throw new IllegalArgumentException("The Dlog group has to have DDH security level");
}
if(!(hash instanceof CollisionResistant)){
    throw new IllegalArgumentException("The hash function has to have CollisionResistant security level");
}

// Everything is correct, then sets the member variables and creates object.
this.dlogGroup = dlogGroup;
qMinusOne = dlogGroup.getOrder().subtract(BigInteger.ONE);
this.hash = hash;
this.random = random;
}
## Results – Average of 1000 Runs

The Cramer-Shoup Encryption Scheme

<table>
<thead>
<tr>
<th>Dlog Group Type</th>
<th>Dlog Provider</th>
<th>Dlog Param</th>
<th>Hash Function</th>
<th>Hash Provider</th>
<th>Encrypt Time (ms)</th>
<th>Decrypt Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DlogZpSafePrime</td>
<td>CryptoPP</td>
<td>1024</td>
<td>SHA-256</td>
<td>BC</td>
<td>6.072</td>
<td>3.665</td>
</tr>
<tr>
<td>DlogZpSafePrime</td>
<td>CryptoPP</td>
<td>2048</td>
<td>SHA-256</td>
<td>BC</td>
<td>43.818</td>
<td>26.289</td>
</tr>
<tr>
<td>DlogECFp</td>
<td>BC</td>
<td>P-224</td>
<td>SHA-1</td>
<td>BC</td>
<td>54.171</td>
<td>31.662</td>
</tr>
<tr>
<td>DlogECF2m</td>
<td>BC</td>
<td>B-233</td>
<td>SHA-1</td>
<td>BC</td>
<td>107.316</td>
<td>65.185</td>
</tr>
<tr>
<td>DlogECF2m</td>
<td>BC</td>
<td>K-233</td>
<td>SHA-1</td>
<td>BC</td>
<td>25.292</td>
<td>14.886</td>
</tr>
<tr>
<td>DlogECFp</td>
<td>Miracl</td>
<td>P-224</td>
<td>SHA-1</td>
<td>BC</td>
<td>6.571</td>
<td>3.929</td>
</tr>
<tr>
<td>DlogECF2m</td>
<td>Miracl</td>
<td>B-233</td>
<td>SHA-1</td>
<td>BC</td>
<td>5.819</td>
<td>3.652</td>
</tr>
<tr>
<td>DlogECF2m</td>
<td>Miracl</td>
<td>K-233</td>
<td>SHA-1</td>
<td>BC</td>
<td>2.753</td>
<td>1.787</td>
</tr>
</tbody>
</table>