SCAPI

The Secure Computation Application Programming Interface

http://crypto.biu.ac.il/about-scapi.php

Yehuda Lindell

Bar-Ilan University

August 20, 2013
CRYPTO 2013 Rump Session

Secure Computation in Practice

- For the last 2.5 decades, secure computation has been a foundational theoretical topic of study

Secure Computation in Practice

- For the last 2.5 decades, secure computation has been a foundational theoretical topic of study
- Recently, interest has grown with respect to the practicality of secure computation
  - Governments, security organizations, industry,...

Secure Computation in Practice

- For the last 2.5 decades, secure computation has been a foundational theoretical topic of study
- Recently, interest has grown with respect to the practicality of secure computation
  - Governments, security organizations, industry,...
- In the last 5 years there has been incredible progress on making secure computation practical
  - Today we can run semi-honest secure computation for problems like secure AES in a quarter of a second
  - Protocols for malicious adversaries exist that give amazing amortized complexity
  - Every year there are new significant breakthroughs

Secure Computation in Practice

- For the last 2.5 decades, secure computation has been a foundational theoretical topic of study
- Recently, interest has grown with respect to the practicality of secure computation
  - Governments, security organizations, industry,...
- In the last 5 years there has been incredible progress on making secure computation practical
  - Today we can run semi-honest secure computation for problems like secure AES in a quarter of a second
  - Protocols for malicious adversaries exist that give amazing amortized complexity
  - Every year there are new significant breakthroughs
- This is very surprising (and exciting): we now know that secure computation can be practical for a reasonably wide range of problems

Implementation of Secure Computation

- 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.)

Basic Design Decisions

- 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

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

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

Design Principles

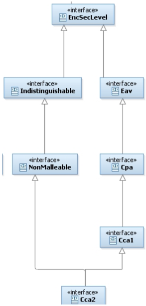
- **Flexibility:**
  - Cryptographers write protocols in abstract terms (OT, commitment, PRF, etc.)
  - SCAPI encourages implementation at this abstract level (work with any "DLOG group" and afterwards instantiate with concrete group and concrete library; e.g. EC-group from Miracl)
  - Can work at many different levels of abstraction, as desired
- **Extendibility:** can add support for any new libraries and implementation by providing wrappers that implement the defined interfaces
- **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

Security Levels

- 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.

Security Levels

- 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:



Layers and Primitives

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, sigma protocols, ZK, ZKPOK, commitments, etc.)

Example Usage

The Cramer-Shoup Encryption Scheme

```
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;
}
```

Example Usage

The Cramer-Shoup Encryption Scheme

```
public AsymmetricCiphertext encrypt(Plaintext plaintext){
  /* Choose a random  r in Zq; calculate u1 = g1^r, u2 = g2^r, e  = (h^r)*msgEl
  * Convert u1, u2, e to byte[] using the dlogGroup
  * Compute alpha  - the result of computing the hash function on the concatenation u1+u2+e.
  * Calculate v = c^r * d^(r*alpha)
  * Create and return an CramerShoupCiphertext object with u1, u2, e and v. */

 ...

  GroupElement msgElement = ((GroupElementPlaintext) plaintext).getElement();

  BigInteger r = chooseRandomR();      //Choose a random value between 0 and q-1 (q = group order)
  GroupElement u1 = calcU1(r);         //Does: dlogGroup.exponentiate(publicKey.getGenerator1(), r);
  GroupElement u2 = calcU2(r);         //Does: dlogGroup.exponentiate(publicKey.getGenerator(), r);
  GroupElement hExpr = calcHExpR(r);   //Does: dlogGroup.exponentiate(publicKey.getH(), r);
  GroupElement e = dlogGroup.multiplyGroupElements(hExpr, msgElement);

  byte[] u1ToByteArray = dlogGroup.mapAnyGroupElementToByteArray(u1);
  byte[] u2ToByteArray = dlogGroup.mapAnyGroupElementToByteArray(u2);
  byte[] eToByteArray = dlogGroup.mapAnyGroupElementToByteArray(e);

  //Calculates the hash(u1 + u2 + e).
  byte[] alpha = calcAlpha(u1ToByteArray, u2ToByteArray, eToByteArray);

  GroupElement v = calcV(r, alpha);    //Calculates v = c^r * d^(r*alpha).

  //Creates and return an CramerShoupCiphertext object with u1, u2, e and v.
  CramerShoupOnGroupElementCiphertext cipher = new CramerShoupOnGroupElementCiphertext(u1, u2, e, v);
  return cipher;
}
```

Example Usage

The Cramer-Shoup Encryption Scheme

```
public static void main(String[] args) throws FactoriesException {
   ...
   // Get parameters from config file:
   CramerShoupTestConfig[] config = readConfigFile();
   ...
   for (int i = 0; i < config.length; i++) {
      result = runTest(config[i]);
      out.println(result);
      System.out.println(result);
   }
   ...
}
```

Example from configuration file:

```
dlogGroup = DlogZpSafePrime
dlogProvider = CryptoPP
algorithmParameterSpec = 1024
hash = SHA-256
providerHash = BC
numTimesToEnc = 1000

dlogGroup = DlogECFp
dlogProvider = BC
algorithmParameterSpec = P-224
hash = SHA-1
providerHash = BC
numTimesToEnc = 1000

dlogGroup = DlogECFp
dlogProvider = Miracl
algorithmParameterSpec = P-224
hash = SHA-1
```

Example Usage

The Cramer-Shoup Encryption Scheme

```
static public String runTest(CramerShoupTestConfig config) throws FactoriesException{
   DlogGroup dlogGroup;
   //Create the requested Dlog Group object. Do this via the factory.
   //If no provider specified, take the SCAPI-defined default provider.
   if(config.dlogProvider != null){
      dlogGroup = DlogGroupFactory.getInstance().getObject(config.dlogGroup+
                              "("+config.algorithmParameterSpec+")", config.dlogProvider);
   }else {
      dlogGroup = DlogGroupFactory.getInstance().getObject(config.dlogGroup+
                              "("+config.algorithmParameterSpec+")");
   }

   CryptographicHash hash;
   //Create the requested hash. Do this via the factory.
   if(config.hashProvider != null){
      hash = CryptographicHashFactory.getInstance().getObject(config.hash, config.hashProvider);
   }else {
      hash = CryptographicHashFactory.getInstance().getObject(config.hash);
   }

   //Create a random group element. This element will be encrypted several times as specified in
   //config file and decrypted several times
   GroupElement gEl = dlogGroup.createRandomElement();

   //Create a Cramer Shoup Encryption/Decryption object. Do this directly by calling the relevant
   //constructor. (Can be done instead via the factory).
   ScCramerShoupDDHOnGroupElement enc = new ScCramerShoupDDHOnGroupElement(dlogGroup, hash);
```

Example Usage

The Cramer-Shoup Encryption Scheme

```
//Generate and set a suitable key.
KeyPair keyPair = enc.generateKey();
try {
   enc.setKey(keyPair.getPublic(),keyPair.getPrivate());
}  catch (InvalidKeyException e) {
   e.printStackTrace();
}

//Wrap the group element we want to encrypt with a Plaintext object.
Plaintext plainText = new GroupElementPlaintext(gEl);
AsymmetricCiphertext cipher = null;

//Measure the time it takes to encrypt each time. Calculate and output the average running time.
long allTimes = 0;
long start = System.currentTimeMillis();
long stop = 0;
long duration = 0;

int encTestTimes = new Integer(config.numTimesToEnc).intValue();
for(int i = 0; i < encTestTimes; i++){
   cipher = enc.encrypt(plainText);
   stop = System.currentTimeMillis();
   duration = stop - start;
   start = stop;
   allTimes += duration;
}
double encAvgTime = (double)allTimes/(double)encTestTimes;

//Repeat for decryption...
```

Results – Average of 1000 Runs

The Cramer-Shoup Encryption Scheme

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