SCAPI

The Secure Computation Application Programming Interface
https://github.com/cryptobiu/scapi

Yehuda Lindell

Software Team: Moriya Farbstein and Meital Levy

Bar-Ilan University

Thursday February 19, 2015
5th BIU Winter School

Implementing Secure Computation

- A typical protocol uses:
  - Oblivious transfer
  - Commitments
  - Zero knowledge
  - Circuits
  - And more...

  Implementing a protocol (well) is a very big project

Implementing Secure Computation

- A typical protocol uses:
  - Oblivious transfer
  - Commitments
  - Zero knowledge
  - Circuits
  - And more...

  Implementing a protocol (well) is a very big project
- There exist general-purpose cryptographic libraries (cryptopp, OpenSSL, BouncyCastle,...) but they are focused on **secure communication**

Implementing Secure Computation

- A typical protocol uses:
  - Oblivious transfer
  - Commitments
  - Zero knowledge
  - Circuits
  - And more...

  Implementing a protocol (well) is a very big project
- There exist general-purpose cryptographic libraries (cryptopp, OpenSSL, BouncyCastle,...) but they are focused on **secure communication**
- There are libraries for secure computation, but are mostly either:
  - Not open source
  - Not maintained and supported
  - Suitable for quick prototyping

Implementation of Secure Computation

- Most academic implementation projects are aimed at solving a specific problem
  - More efficiently
  - With better security

Implementation of Secure Computation

- Most academic implementation projects are aimed at solving a specific problem
  - More efficiently
  - With better security
- SCAPI is an implementation project with no specific problem in mind
  - SCAPI is a general-purpose secure computation library (infrastructure)

SCAPI Basics

- An open-source project:
  `https://www.github.com/cryptobiu/scapi`
- Long-term commitment (as long as we have money) to:
  - Provide **support** to SCAPI users
  - Fix bugs
  - Improve existing implementations (efficiency, security)
  - Add functionality: protocols, primitives, etc.

SCAPI Basics

- An open-source project:
  `https://www.github.com/cryptobiu/scapi`
- Long-term commitment (as long as we have money) to:
  - Provide **support** to SCAPI users
  - Fix bugs
  - Improve existing implementations (efficiency, security)
  - Add functionality: protocols, primitives, etc.
- We are happy to receive code contributions

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)

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):
  - OpenSSL
  - Miracl
  - Cryptopp

Design Principle 1 – Flexibility

- Cryptographers write protocols in abstract terms (OT, commitment, PRF, etc.)
- SCAPI encourages implementation at this abstract level

Design Principle 1 – Flexibility

- Cryptographers write protocols in abstract terms (OT, commitment, PRF, etc.)
- SCAPI encourages implementation at this abstract level
- How does it work?
  - SCAPI defines interfaces that represent cryptographic primitives
  - A protocol that uses OT, commitment and a group in which DDH is assumed to be hard receives objects of these types in its constructor
  - The application calling the protocol instantiates the appropriate concrete objects and hands them to the protocol

Design Principle 1 – Flexibility

- Cryptographers write protocols in abstract terms (OT, commitment, PRF, etc.)
- SCAPI encourages implementation at this abstract level
- How does it work?
  - SCAPI defines interfaces that represent cryptographic primitives
  - A protocol that uses OT, commitment and a group in which DDH is assumed to be hard receives objects of these types in its constructor
  - The application calling the protocol instantiates the appropriate concrete objects and hands them to the protocol
  - A protocol can receive
    - Any pseudorandom permutation (using the PRP interface)
    - Any AES implementation (using the AES interface)
    - AES from a specific library

Design Principle 1 – Flexibility

- The protocol code is independent of actual primitives
  - Can easily compare the ramification of using different elliptic curve groups (for example)
  - The same code can run on a mobile device (in Java) and on a PC (using native code via JNI)
    - Don't need to reimplement or suffer the inefficiency of Java-only on a PC
  - Primitives or libraries added later can be utilized by previously-implemented protocols (extendibility and efficiency – next)

On Comparing Primitives

- It may seem that one should always just use the “fastest primitive”, in which case the ability to compare different elliptic curve groups is not really interesting

On Comparing Primitives

- It may seem that one should always just use the "fastest primitive", in which case the ability to compare different elliptic curve groups is not really interesting
- But not all primitives are comparable in this way:

On Comparing Primitives

- It may seem that one should always just use the "fastest primitive", in which case the ability to compare different elliptic curve groups is not really interesting
- But not all primitives are comparable in this way:
  - Some are based on less established assumptions (if they are much faster, then maybe it's worth it, but if they only improve the overall time by a little, then maybe not)

On Comparing Primitives

- It may seem that one should always just use the "fastest primitive", in which case the ability to compare different elliptic curve groups is not really interesting
- But not all primitives are comparable in this way:
  - Some are based on less established assumptions (if they are much faster, then maybe it's worth it, but if they only improve the overall time by a little, then maybe not)
  - Some are better for some operations and worse for others: Koblitz curves are faster for regular multiplications, but are slower when a fixed base is used
    - In a protocol where regular exponentiations are mixed with fixed-base exponentiations, it's not necessarily easy to know what is best, **until you try**...

Design Principle 2 – Extendibility

- SCAPI is a general infrastructure and so it's important that new implementations can be added later
  - Every primitive has an interface
  - Any future implementation of a primitive just needs to implement the interface

Design Principle 2 – Extendibility

Example 1 – Oblivious Transfer

- Seven years ago, OT with security against malicious adversaries was horribly inefficient
- We now have highly efficient protocols for this
- Higher level protocols that use OT that were previously implemented need to be changed
  - This change can be trivial, but may also require working over a different type of group altogether and so can involve many changes
- In SCAPI, the new OT can be utilized by all protocols that were implemented at the appropriate level of abstraction

Design Principle 2 – Extendibility

Example 2 – Libraries

- We have incorporated primitives from Bouncy Castle, OpenSSL, Crypto++, and Miracl
- Assume that a new, faster, more secure library for elliptic curve operations is released
  - All that needs to be done is to write a SCAPI wrapper for the library and all existing protocols can take advantage of the new library

Design Principle 3 – Efficiency

- Any infrastructure for secure computation protocols must take efficiency into account
- SCAPI achieves high efficiency via JNI and wrapping fast low-level libraries (the overhead of JNI is very small)

Design Principle 3 – Efficiency

- Any infrastructure for secure computation protocols must take efficiency into account
- SCAPI achieves high efficiency via JNI and wrapping fast low-level libraries (the overhead of JNI is very small)
- There is no doubt that implementing an entire protocol in C and optimizing at a low level will give **better results**
  - But with SCAPI you still get fast implementations that are quicker to implement, modular, suitable for reuse, and so on
- Sometimes, SCAPI wraps a large computation written in native code (garbling, OT extension)

Ease of Use

- Most cryptographic libraries are tailored for encryption and authentication, and **not secure computation**
  - Low-level group operations are typically buried deep down as utilities
  - Libraries don't use the terminology that we are used to
  - Forcing a decision about which concrete implementation to use at the onset is problematic since inefficiencies are often hard to predict

Ease of Use

- Most cryptographic libraries are tailored for encryption and authentication, and **not secure computation**
  - Low-level group operations are typically buried deep down as utilities
  - Libraries don't use the terminology that we are used to
  - Forcing a decision about which concrete implementation to use at the onset is problematic since inefficiencies are often hard to predict
- SCAPI is documented, commented and (hopefully) written clearly – it was written explicitly with other users in mind
  see: `http://scapi.readthedocs.org`

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.
- How does SCAPI differentiate between:
  - A group in which CDH is hard but DDH is not
  - A commitment scheme which is perfectly binding versus perfectly hiding versus something else
  - A hash function which is target collision resistant but not collision resistant

Security Levels

- Consider a protocol that uses any asymmetric encryption scheme that is NM-CCA1 (non-malleable under CCA1 attacks)

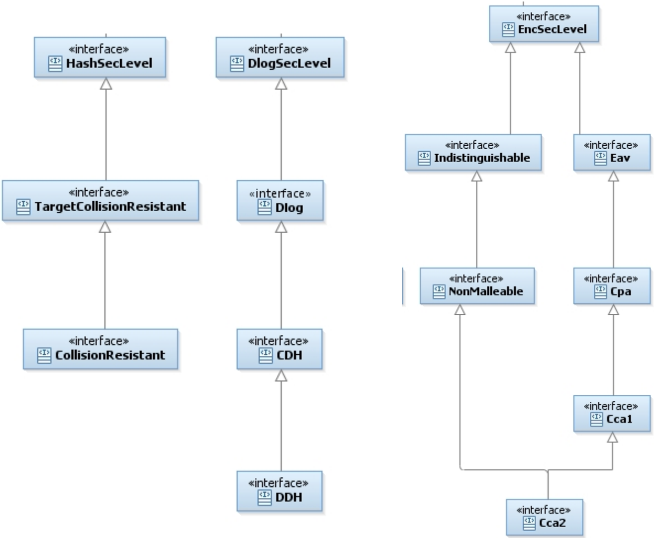
Security Levels

- Consider a protocol that uses any asymmetric encryption scheme that is NM-CCA1 (non-malleable under CCA1 attacks)
  - Can the protocol use Cramer-Shoup (which is CCA2-secure)?

Security Levels

- Consider a protocol that uses any asymmetric encryption scheme that is NM-CCA1 (non-malleable under CCA1 attacks)
  - Can the protocol use Cramer-Shoup (which is CCA2-secure)?
  - If the protocol is written so that it works with any asymmetric encryption scheme, then what happens if it is given a CPA-secure scheme instead?

SCAPI defines **hierarchies of interfaces** for security levels



Security Level Use

- 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 will be defined for protocols (semi-honest, covert, malicious, stand-alone, UC secure, and so on)

Layers and Primitives

SCAPI has three layers

- Basic primitives
- Non-interactive schemes
- Interactive protocols (not in the current release)

Layer 1 – Basic Primitives

- Most of the code at this level is wrappers
  - The exceptions: HKDF, universal hash, Luby-Rackoff, and more
- This layer provides a common interface for low-level libraries
  - Same interface for Bouncy Castle, Crypto++, OpenSSL, Miracl (and whatever else in the future)
- This provides the flexibility and extendibility that we discussed

Layer 1 – Basic Primitives

- Different levels of abstraction
- A protocol can be written using any
  - PRF
  - PRP
  - AES (from any library)
  - AES from a specific library (not a good idea)

Layer 1 – Implemented Primitives

- Pseudorandom functions and permutations
  - Fixed lengths, varying lengths, etc.
- Cryptographic hash functions
- Universal hash functions
- Trapdoor permutations
- Pseudorandom generators
- Key derivation functions
- Discrete log groups
  - This has the most novelty – the same API exists for groups based on $\mathbb{Z}_p*$ and elliptic curves, and for elliptic curves over a prime-order field or a binary field, and for Koblitz curves...

Layer 2 – Non-Interactive Schemes

- Essentially encryption, signatures and MACs
  - Commitments are not included since they are also interactive
- Asymmetric schemes implemented:
  - RSA-OAEP (BC and Crypto++)
  - El Gamal over any dlog group
    - Encryption of group element or byte array (former is important for proving ZK statements about the ciphertext)
  - Cramer-Shoup over any dlog group
    - As above, encryption of group element or byte array
  - Damgård-Jurik
- Other standard schemes: AES with CBC or CTR, CBC-MAC, DSA and RSA signatures, and so on

Layer 3 – Interactive Protocols

- Sigma protocols
  - Over 10 common protocols (DLOG, DDH, Jurik-Damgård and more)
  - Operations: AND of multiple statements, OR or two or more statements, transformation to ZK and ZKPOK, Fiat-Shamir to NIZK, transformation to UCZK
- Commitments
  - Pedersen, ElGamal, hash-based, equivocal, extractable, fully trapdoor, homomorphic, non-malleable, UC

Layer 3 – Interactive Protocols

- Oblivious transfer
  - Semi-honest
    - Stand-alone (Naor-Pinkas optimized)
    - OT extension (ACM CCS 2013 version)
  - Malicious
    - Privacy only
    - One-sided simulation
    - Full simulation – stand-alone
    - UC secure
    - OT extension (to be added soon)
- Garbled circuits
  - Basic and optimized (free XOR, fixed AES, etc.)
- Coin tossing (single bit, string, semi-simulatable)

Layer 3 – Interactive Protocols

Plans for the future:

- Improvements on existing protocols
- Adding new functionality
- Improving overall infrastructure (e.g., the communication layer was just improved to add Queue functionality as well as Socket)

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

Garbled Circuit Example

```java
public void fastCircuitExample() throws NotAllInputsSetException, InvalidKeyException, CheatAttemptException{
    SecureRandom random = new SecureRandom();

    //Prepare a seed to use when garbling.
    byte[] seed = new byte[16];
    random.nextBytes(seed);

    //Create a circuit with Free XOr and without row reduction.
    ScNativeGarbledBooleanCircuit fastGarbledCircuit = new ScNativeGarbledBooleanCircuit("AES_Final-2.txt", true, false);

    //Garble the circuit.
    FastCircuitCreationValues initialValues = fastGarbledCircuit.garble(seed);

    //Set inputs.
    byte[] inputKeys = setInputForFast(fastGarbledCircuit, initialValues);
    fastGarbledCircuit.setInputs(inputKeys);
    //Compute the circuit.

    byte[] outputKeys = fastGarbledCircuit.compute();

    //Translate the garbled output to meaningful output.
    fastGarbledCircuit.translate(outputKeys);

}
```

Zero Knowledge Prover Example

```java
public void exampleZKFromSigmaProtocol(Channel channel) throws IOException, CheatAttemptException,
                                    ClassNotFoundException, CommitValueException {
    //Create the necessary parameters for the prover.
    DlogGroup dlog = new OpenSSLDlogECF2m("K-233");
    int t = 80;
    SecureRandom random = new SecureRandom();
    //Create sigma prover computation.
    SigmaProverComputation sigmaProver = new SigmaDHProverComputation(dlog, t, random);

    ZKProver prover = null;
    try {
        //Create the commitment receiver used in the protocol.
        CmtReceiver ctReceiver = new CmtPedersenReceiver(channel, dlog, random);
        //Create the ZK prover.
        prover = new ZKFromSigmaProver(channel, sigmaProver, ctReceiver);
    } catch (SecurityLevelException e) {
        // Should not occur since the given Dlog has the necessary Security level.
    } catch (InvalidDlogGroupException e) {
        // Should not occur since the given Dlog is valid.
    }

    SigmaProverInput input = createInput(dlog);

    //Prove.
    prover.prove(input);
}
```

Zero Knowledge Verifier Example

```
public void exampleZKFromSigmaProtocol(Channel channel) throws IOException, CheatAttemptException,
                                                    ClassNotFoundException, CommitValueException {
    //Create the necessary parameters for the verifier.
    DlogGroup dlog = new OpenSSLDlogECF2m("K-233");
    int t = 80;
    SecureRandom random = new SecureRandom();

    ZKVerifier verifier = null;
    try {
        //Create sigma verifier computation.
        SigmaVerifierComputation sigmaVerifier = new SigmaDHVerifierComputation(dlog, t, random);
        //Create the commitment's committer used in the protocol.
        CmtCommitter ctCommitter = new CmtPedersenCommitter(channel, dlog, random);
        //Create the ZK verifier.
        verifier = new ZKFromSigmaVerifier(channel, sigmaVerifier, ctCommitter, random);
    } catch (SecurityLevelException e) {
        // Should not occur since the given Dlog has the necessary Security level.
    } catch (InvalidDlogGroupException e) {
        // Should not occur since the given Dlog is valid.
    }

    //Create the inputs for the verifier.
    SigmaCommonInput input = createInput(dlog);

    //Verify and print the result.
    boolean verified = verifier.verify(input);
    System.out.println("verification result = " + verified);
}
```

Proving Different Languages

Replace:

```
//Create sigma verifier computation.
SigmaVerifierComputation sigmaVerifier = new SigmaDHVerifierComputation(dlog, t, random);
```

with:

```
//Create sigma verifier computation.
SigmaVerifierComputation sigmaVerifier = new SigmaDHExtendedVerifierComputation(dlog, t, random);
```

Summary

- SCAPI is an open-source library for secure computation implementations
- Currently, the focus is on primitives for the no honest-majority setting (the vision is to add honest-majority tools as well)
- We plan on supporting SCAPI in the long term
  - Help to users
  - Bug fixes
  - Improve existing code
  - Expand code base