EasyCrypt - Hashed ElGamal semantic security proof exercise

Vitor Pereira

FMiTF Bootcamp - May 29 - June 2, 2023

1 Introduction

The purpose of this document is to describe a verification exercise that consists on performing the security proof of the ElGamal encryption scheme in its hashed version. We will first give an overview of how such proofs are done in EasyCrypt using the non-hashed ElGamal cryptosystem as an example.

We will closely follow the proof provided at [1], where the game hopping proof technique is also explained. The student is encouraged to closely follow [1] while reading this document in order to match the building blocks described here with the proof described in [1].

2 Modeling public key encryption Schemes in EasyCrypt

A public key encryption scheme is a 3-tuple of algorithms:

- KenGen() : pk * sk - a probabilistic algorithm that generates both the public and secret keys (pk and sk, respectively).
- Encrypt(pk, pt) : ct - a probabilistic algorithm that takes as input the public key pk and the plaintext pt and outputs a ciphertext ct.
- Decrypt(sk, ct) : pt - a deterministic algorithm that takes as input the secret key sk and the ciphertext ct and outputs a plaintext pt.

In EasyCrypt, the modeling of public key encryption schemes can be captured by a `module type`, as follows

```
type pk_t.
type sk_t.
type pt_t.
type ct_t.

module type Scheme = {
  proc key_gen() : pk_t * sk_t
  proc encrypt(pk : pk_t, m : pt_t) : ct_t
  proc decrypt(sk : sk_t, ct : ct_t) : pt_t
}.
```

3 Modeling security definitions in EasyCrypt

Cryptographic semantic security, also dubbed as *indistinguishability against chosen-plaintext attacks* or simply *IND-CPA*, is a security definition that is captured by the following security experience (or *game*):

1. First, an *adversary* (an entity trying to break the *IND-CPA* security), chooses two messages, say m_0 and m_1.
2. Next, the *challenger* selects a random bit b and encrypts message m_0 if $b = false$ or encrypts m_1 if $b = 1$.
3. The *adversary* is then given the encryption of either m_0 or m_1

4. Finally, the *adversary* must output a *decision* bit b': it needs to determine if the ciphertext it got from the *challenger* corresponds to an encryption of m_0 or m_1.

If the scheme is *IND-CPA* secure, then the *adversary* is said to have no *advantage* against the scheme and its best option is to *guess* which message originated the ciphertext it received. Formally, a scheme is *IND-CPA* secure if the probability that the *adversary* outputs $b' = b$ is $\frac{1}{2}$.

In EasyCrypt, the *IND-CPA* security game can be defined as a module as follows

```
module type INDCPA_Adv = {
  proc gen_query(pk : pk_t) : pt_t * pt_t
  proc guess(ct : ct_t) : bool
}.

module INDCPA (S : Scheme) (A : INDCPA_Adv) = {
  proc main() : bool = {
    var pk, sk, m0, m1, b, b', ct;

    (pk, sk) <@ S.key_gen();
    (m0, m1) <@ A.gen_query(pk);
    b <$ {0,1};
    ct <@ S.encrypt(pk, if b then m1 else m0);
    b' <@ A.guess(ct);

    return (b = b');
  }
}.
```

4 EasyCrypt ElGamal specification

We first provide a brief description of the ElGamal cryptosystem. Let G be a group of prime order q, and let $g \in G$ be a generator. The key generation algorithm computes (pk, sk) as follows

$$x \xleftarrow{\$} \mathbb{Z}_q; pk \leftarrow g^x; sk \leftarrow x$$

To encrypt a message $m \in G$, the encryption algorithm proceeds as follows

$$y \xleftarrow{\$} \mathbb{Z}_q; \beta \leftarrow g^y; \delta \leftarrow pk^y; \zeta \leftarrow \delta \cdot m; ct \leftarrow (\delta, \zeta)$$

Finally, decryption of a ciphertext is simply done by

$$m < -\zeta / \beta^{sk}$$

The ElGamal encryption scheme can be defined in EasyCrypt as a module of type `Scheme` in accordance to the following code.

```
module ElGamal : Scheme = {
  proc key_gen() : pk_t * sk_t = {
    var pk, sk;

    sk <$ FDistr.dt;
    pk <- g ^ sk;

    return (pk, sk);
  }

  proc encrypt(pk : pk_t, m : pt_t) : ct_t = {
    var y, bet, delt, zet;

    y <$ FDistr.dt;
    bet <- g ^ y;
    delt <- pk ^ y;
    zet <- delt * m;

    return (bet, zet);
  }

  proc decrypt(sk : sk_t, ct : ct_t) : pt_t = {
    var m, bet, zet;

    (bet, zet) <- ct;
```

```
    m <- zet / (bet ^ sk);

    return (m);
  }
}.
```

where `FDistr.dt` is a uniform probability distribution over elements of the finite field $\mathbb{Z}_q$.

Finite field and cyclic group arithmetic libraries are provided in the EasyCrypt scripts that accompany this document.

5 ElGamal semantic security proof

ElGamal encryption is semantically secure under the Decisional Diffie-Hellman (DDH) assumption. This is the assumption that it is hard to distinguish triples of the form (g^x, g^y, g^{x*y}) from triples of the form (g^x, g^y, g^z), where x, y, and z are random elements of $\mathbb{Z}_q$.

5.1 DDH assumption in EasyCrypt

The DDH assumption is formulated in EasyCrypt as follows

```
module type Adversary = {
  proc guess(gx gy gz : group) : bool
}.

module DDH0 (A : Adversary) = {
  proc main() : bool = {
    var b, x, y;

    x <$ FDistr.dt;
    y <$ FDistr.dt;
    b <@ A.guess(g ^ x, g ^ y, g ^ (x * y));
    return b;
  }
}.

module DDH1 (A : Adversary) = {
  proc main() : bool = {
    var b, x, y, z;

    x <$ FDistr.dt;
    y <$ FDistr.dt;
    z <$ FDistr.dt;
    b <@ A.guess(g ^ x, g ^ y, g ^ z);
    return b;
  }
}.
```

where the *advantage* against the DDH is formulated as

$$|Pr[DDH0(A).main : b = 1] - Pr[DDH1(A).main : b = 1]|$$

5.2 DDH reduction proof

Cryptographic reduction proof are done following a *contradiction* approach. For example, to reduce the security of the ElGamal encryption scheme to the DDH assumption, one needs to prove that any procedure that can be used to break the security of the scheme can also be used to break the DDH assumption. However, DDH is considered a hard mathematical problem and, therefore, no algorithm can be used to solve it in computational time, meaning that we arrived at a contradiction. Consequently, there is no *adversary* is able to successfully break the security of ElGamal, since that would mean that the DDH assumption would not be hard problem, as it is assumed.

To perform the reduction proof in EasyCrypt, one needs to write a DDH *adversary* that uses the *IND-CPA* adversary as a sub-routine. That adversary is written bellow.

```
module D (A : INDCPA_Adv) = {
  proc guess(gx gy gz : group) : bool = {
    var m0, m1, b, b';
```

```
    (m0, m1) <@ A.gen_query(gx);
    b <$ {0,1};
    b' <@ A.guess(gy, gz * (if b then m1 else m0));

    return (b = b');
  }
}.
```

The `D` *adversary* first queries `A` (the *IND-CPA adversary*) to generate two messages and then uses `A guess` procedure (the procedure used by `A` to break ElGamal's security) to procedure its own decision bit.

Next, we define a new *game* that is equivalent to the original *IND-CPA* experience, except that the value ζ is calculated as $\zeta \leftarrow g^z$, with random $z \in \mathbb{Z}_q$, instead of $\zeta \leftarrow \delta \cdot m$.

```
module Game1 = {
  proc main() : bool = {
    var pk, sk, m0, m1, b, b', ct, y, bet, delt, z, zet;

    (pk, sk) <@ ElGamal.key_gen();
    (m0, m1) <@ A.gen_query(pk);
    b <$ {0,1};

    y <$ FDistr.dt;
    bet <- g ^ y;
    delt <- pk ^ y;
    z <$ FDistr.dt;
    zet <- g ^ z;
    ct <- (bet, zet);

    b' <@ A.guess(ct);

    return (b = b');
  }
}.
```

Now, we are able to prove that the `D` *adversary* interpolates both the *IND-CPA game* (which is typically called *Game0*) and `Game1`. That is done by proving the following two equivalence lemmas

```
lemma game0_ddh0_equiv : equiv [ DDH0(D(A)).main ~ Game0(A).main : ={glob A} ==> ={res} ].

lemma game1_ddh1_equiv : equiv [ DDH1(D(A)).main ~ Game1.main : ={glob A} ==> ={res} ].
```

and, since both games are equivalent, they output the same bit with equivalent probability

```
lemma game0_ddh0_pr &m : Pr [ DDH0(D(A)).main() @ &m : res ] = Pr [ Game0(A).main() @ &m : res ].

lemma game1_ddh1_pr &m : Pr [ DDH1(D(A)).main() @ &m : res ] = Pr [ Game1.main() @ &m : res ].
```

The complete proof of these lemmas can be found in the accompanying EasyCrypt files.

Note that the output of `Game1` (`b = b'`) is independent of the bit `b`, since no message is actually encrypted. Therefore, one can prove that the probability of the event `b = b'` is precisely $\frac{1}{2}$.

```
lemma game1_pr &m : Pr [ Game1.main() @ &m : res ] = 1%r / 2%r.
```

Finally, we can establish the concrete security bounds of the ElGamal encryption scheme by proving the following lemma

```
lemma security &m :
  Pr [ INDCPA(ElGamal, A).main() @ &m : res ] =
    1%r/2%r + (Pr[ DDH0(D(A)).main() @ &m : res ] - Pr[ DDH1(D(A)).main() @ &m : res ]).
```

where `Pr[ DDH0(D(A)).main()@ &m : res ] - Pr[ DDH1(D(A)).main()@ &m : res ]` represents the (negligible) advantage against the DDH assumption.

6 Exercise: semantic security proof of hashed ElGamal

Your goal is to take advantage of the description of the semantic security proof of ElGamal described above and detailed in the EasyCrypt files that are attached to this document, and perform the semantic security of the ElGamal encryption scheme in its hashed version.

Briefly, the security of hashed ElGamal can be reduced to the DDH assumption and to the *entropy smoothing* assumption of the underlying hash function. The first reduction step is the same of the non-hashed ElGamal version. You will need to do the second reduction proof, i.e., the reduction to the *entropy smoothing* hash function assumption. You will not need to formalize this assumption, as it is part of the EasyCrypt set of files available to you. Similarly to the DDH reduction proof, you will need to write an *entropy smoothing* adversary based on an *IND-CPA* adversary and write a new game hop that interpolates with the this new assumption.

The complete security proof of the hashed ElGamal cryptosystem can also be found at [1].

References

[1] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Paper 2004/332, 2004. `https://eprint.iacr.org/2004/332`.