Implementing transformations based on discrete logarithm for ProveIt

Research Seminar in Cryptography

Cyber Security Master Programme

University of Tartu

Yauhen Yakimenka
Supervised by Kristjan Krips & Sven Laur

Abstract

ProveIt is the semi-automatic tool for step-by-step security proofs of cryptographic primitives. The tool is given the formalisation of a security property through a short program also called a game. Proofs are made via a sequence of transformations. The aim of the seminar's project is to implement transformations based on discrete logarithm.

Contents

1 Introduction

Systems in information security usually use cryptographic primitives such as hash functions, MACs, signatures, etc. When more than one party is involved, such an interaction between parties (machines) is called cryptographic protocol. Purpose of cryptographic protocols is establishing security in potential presence of malicious adversaries. Cryptographers job usually involves proving the security properties of the protocol.

The way to write down a cryptographic primitive we consider in this project is a game. The game is a representation of the primitive as a program in some formal language (it could be also some conventional programming language). By changing the game it is possible sometimes to obtain another game, which security properties are known. Such a sequence of games is referred as a game-based proof.

Game-based proofs are more natural way of proving for human beings. Besides automatic proving tools, many proofs are still handled by hand. But although such proofs are conceptually simple, they could be technically complex. ProveIt is a semi-automatic tool aimed to help with these problems.

ProveIt is being developed in Tartu. It allows researchers to make changes to the code-based probabilistic game and does the rewriting automatically in order to guarantee correctness and save the researcher's time. The idea behind ProveIt is to simplify the process of proving a cryptographic protocol in a game-based way and also to keep it understandable.

Figures in this repost have been taken from lecture slides for Cryptology II course by Sven Laur.

1.1 Game-based proofs

Game-based proof is a sequence of games $G, G_1, G_2, \ldots, G^*$ where G models a real world threat and G^* is in some way easier to analyse game. Firstly, the initial game is written in some formal (programming) language. Usually, the game finishes with the output 1 if the attacker was successful. We can find the probability of the game ending with a certain output.

Game G_1 is a (small) modification of game G. It can be either syntactic modification (i.e. syntactic transformation that does not change the output distribution of the adversary) or a cryptographic modification (i.e. cryptographic transformation that can modify the game so that the output distribution of the adversary changes).

G_2 is in turn a modification of game G_1, and so on and so forth. Each step can change the success of an attack. The overall goal is to obtain the game G^*, whose attack success probability is known (or we could bound it). If we know how each transformation changes the security properties of the game, we could obtain the security properties of original game G.

1.2 ProveIt tool

ProveIt works as follows. The user first enters initial game in the ProveIt language and then applies transformations (they are predefined and available as toolbar buttons). For every step ProveIt checks if particular transformation is applicable for the current game/place in the game's code. After every step the

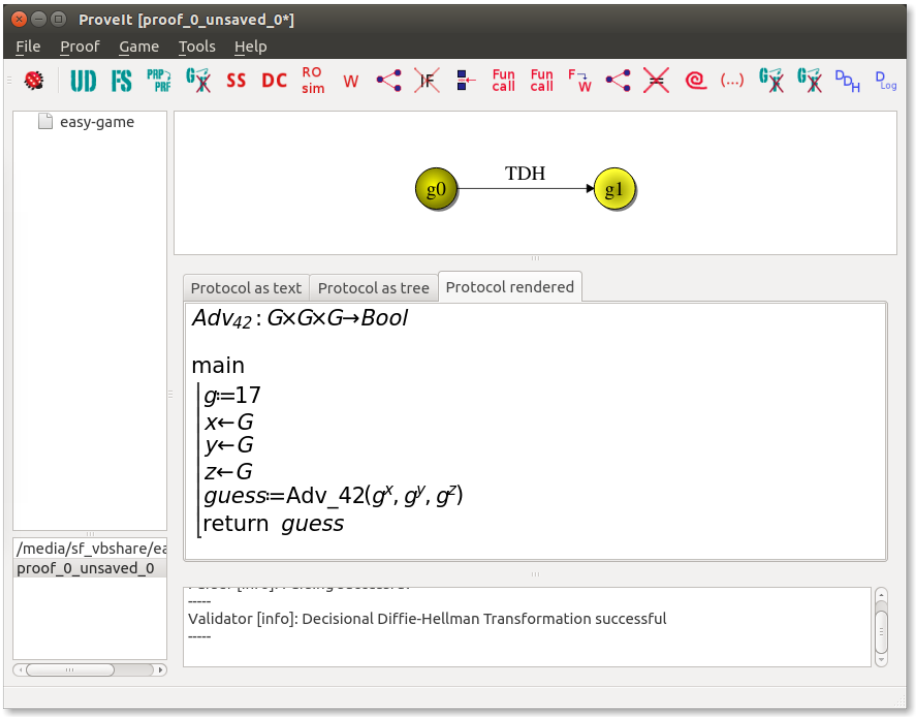

Figure 1: ProveIt interface

game is re-written automatically. Every intermediate game is available, so that user can return to it and try to apply different transformations. The ProveIt interface is shown at Figure 1.

Now we briefly introduce the main features of ProveIt language. The ProveIt language consists of statements – assignments, uniform choice, functions signatures, the main game definition, function definition, typing statements and control-flow statements.

Functions must have a signature before the definition, specifying the name of the function, parameters' types and the type of the return value. The function definition is a body of the function with return statement.

Assignment gives a value to a variable on the left-hand side. Uniform choice uniformly choose a value from the given set on the right-hand side and gives it to a variable on the left-hand side. Control-flow statements are if statements, (do-)while and for loops.

Basically, most of the constructions of the language are intuitively understandable for a person acquainted with some C-like programming languages.

2 Transformations based on DL

2.1 DL problem

The discrete logarithm (DL) is the inverse of discrete exponentiation in a finite cyclic group. Given a cyclic group G with group operation $\cdot$ and a generator g,

exponentiation in G is defined by

$$g^x = \underbrace{g \cdot g \cdot \ldots \cdot g}_{x \text{ times}}$$

Suppose $y = g^x$, then the discrete logarithm of y is x and is written as

$$\log_g y = x$$

Actually, the discrete logarithm of y is not unique as it can only be found modulo the order of g in G.

Discrete exponentiation within a group can be performed quickly, doing only $O(\log x)$ group operations, by using the method of fast exponentiation; however, discrete logarithms appear to be much harder to compute. All methods for computing discrete logarithms designed to work in all cyclic groups require exponential time, with the fastest requiring $O(\sqrt{n})$.

2.2 Diffie-Hellman protocol

The problem of computing discrete logarithms was just a mathematical curiosity until, in 1976, Diffie and Hellman described a method of exchanging cryptographic keys which relies on the hardness of the discrete logarithm problem for its security [DH76]. The key exchange between two parties, A and B, works as follows:

1. A and B agree on group G and generator g. These choices can be public.
2. A chooses an exponent a at random, computes g^a, and sends this value to B. The exponent a must be kept secret.
3. B chooses an exponent b at random, computes g^b, and sends this values to A. The exponent b must be kept secret. B then computes, using the value g^a received from A, $K_b = (g^a)^b$.
4. When A receives g^b from B, A computes $K_a = (g^b)^a$.

If all goes well, it should be the case that $K_a = K_b$ and that this key is only known by A and B. These two parties can now use this private key to communicate using some cryptographically secure communication protocol.

During this key exchange protocol, the values g^a and g^b are publicly visible. The Diffie-Hellman conjecture states that it is computationally difficult to compute g^{ab} from the known values of g^a and g^b. Clearly, if discrete logarithms were easy to compute, then one could simply compute $b = \log_g(g^b)$ and then compute $g^{ab} = (g^a)^b$. It is not known if there is an easy way to compute g^{ab} from knowledge of g^a and g^b without computing the discrete logarithms a and b.

2.3 Plain discrete logarithm transformation

Let's say we have the following game:

$$
\begin{aligned}
&Adv\colon G \times G \to G \\
&\mathcal{G}_0 \\
&\left[\begin{array}{l} g := 17 \\ x \leftarrow \mathbb{Z}_{|G|} \\ \text{return } x = Adv(g, g^x) \end{array}\right.
\end{aligned}
$$

Assume that G is secure, then from (G, g, g^x) adversary obtains x with negligible probability only. More precisely, we assume that:

1. The group G and generator g are secure
2. The adversary does not have any direct or indirect access to x (except that via g^x)

Then we could re-write the game in the following way:

$$
\begin{aligned}
&Adv\colon G \times G \to G \\
&\mathcal{G}_1 \\
&\left[\begin{array}{l} g := 17 \\ x \leftarrow \mathbb{Z}_{|G|} \\ \gamma \leftarrow G \\ \text{return } x = Adv(g, \gamma) \end{array}\right.
\end{aligned}
$$

From DL assumption we deduce that

$$|\mathrm{P}\{\mathcal{G}_0 = 1\} - \mathrm{P}\{\mathcal{G}_1 = 1\}| < \text{negligible}.$$

2.4 Decisional Diffie-Hellman transformation

Decisional Diffie-Hellman problem is illustrated at the following figure:

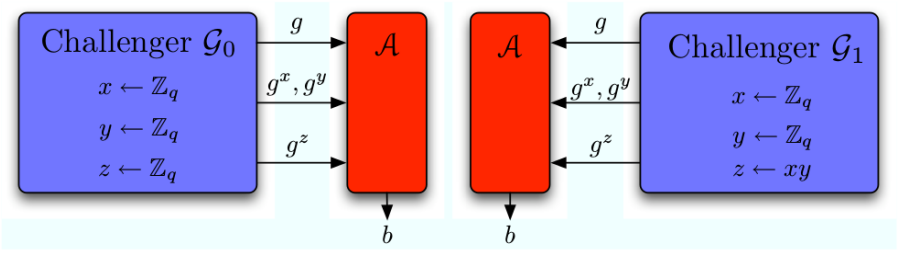

Figure 2: Decisional Diffie-Hellman for group of order q

The aim of the adversary is to distinguish the situations when the challenger has sent randomly chosen z or it is a product of x and y indeed. Therefore for secure group G the adversary, which does not have any information about

neither x not y (except via g^x and g^y), does not distinguish $\mathcal{G}_0$ and $\mathcal{G}_1$. This is used as a basis for the following transformation.

Consider the following game:

$$
\begin{array}{l}
Adv\colon G \times G \times G \times G \to Bool \\
\mathcal{G}_0 \\
\left[\begin{array}{l}
g := 17 \\
x \leftarrow \mathbb{Z}_{|G|} \\
y \leftarrow \mathbb{Z}_{|G|} \\
z := xy \\
\text{return } Adv(g, g^x, g^y, g^z)
\end{array}\right.
\end{array}
$$

At the moment generator g is requested from the user via dialogue window.

The transformation is based on the idea that for adversary z will look like a random, therefore we could re-write the game as follows:

$$
\begin{array}{l}
Adv\colon G \times G \times G \times G \to Bool \\
\mathcal{G}_1 \\
\left[\begin{array}{l}
g := 17 \\
x \leftarrow \mathbb{Z}_{|G|} \\
y \leftarrow \mathbb{Z}_{|G|} \\
z \leftarrow \mathbb{Z}_{|G|} \\
\text{return } Adv(g, g^x, g^y, g^z)
\end{array}\right.
\end{array}
$$

The aforementioned transformation could be applied if:

- The group G is secure enough. This part relies on decision of user.
- x and y should be defined as stated above, i.e. by uniform choice from the same group. This applies to the last definition before z is defined as multiplication of x and y. After that x and y should not change.
- Values of x, y and z should not be leaked in other places of the game. This requires dependency checks.

Under aforementioned assumptions and conditions the difference of success probability of adversary in $\mathcal{G}_0$ and $\mathcal{G}_1$ is negligible.

2.5 Computational Diffie-Hellman transformation

Computational Diffie-Hellman problem is illustrated in the following figure:

Consider the following game:

$$
\begin{array}{l}
Adv\colon G \times G \times G \to G \\
\mathcal{G}_0 \\
\left[\begin{array}{l}
g := 17 \\
x \leftarrow \mathbb{Z}_{|G|} \\
y \leftarrow \mathbb{Z}_{|G|} \\
\text{return } g^{xy} = Adv(g, g^x, g^y)
\end{array}\right.
\end{array}
$$

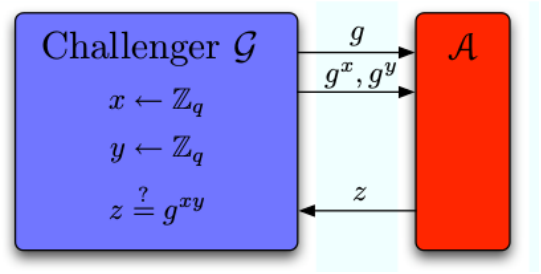

Figure 3: Decisional Diffie-Hellman for group of order q

If we consider the group G secure then from g^x and g^y adversary obtains no information on x and y. We could re-write the game in the following way:

$$
\begin{aligned}
&Adv\colon G \times G \times G \to G \\
&\mathcal{G}_0 \\
&\left[\begin{array}{l}
g := 17 \\
x \leftarrow \mathbb{Z}_{|G|} \\
\chi \leftarrow G \\
y \leftarrow \mathbb{Z}_{|G|} \\
\gamma \leftarrow G \\
\text{return } g^{xy} = Adv(g, \chi, \gamma)
\end{array}\right.
\end{aligned}
$$

3 Conclusion

At the moment most of things are still work in progress. At the moment only DL and DDH transformations have been implemented.

References

[DH76] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.