Logic Synthesis based Public Key Scheme

Boaz Shahar
boaz.public.123@gmail.com

Abstract. This article proposes a method for the construction of a public key system that is based on VLSI logic synthesis algorithms. First, we discuss the properties of VLSI logic synthesis algorithms. Then we view them in the context of cryptographic primitives. Then we propose a public key encryption system and finally discuss its security properties.

1 Introduction

One drawback of a symmetric key encryption schemes is that it requires a priori communication of the key between A and B using a secure channel.

The development of ***Public Key Cryptography*** in the 20'th century enables dropping this requirement. The receiver B can publish an information item called a *Public Key* for any one including the sender A and any potential adversary. Anyone who has the public-key can encrypt a message. The receiver B keeps secret information for himself alone, called the *Private Key*. The private key enables the receiver to decrypt the message encrypted by anyone using the public key.

A public key encryption scheme can be constructed given a *trapdoor function*. A trapdoor function is a one way function for which there exists some trapdoor secret information, known to the receiver alone, with which the receiver can invert the function.

The idea of public key cryptography and trapdoor functions was presented first by Diffie and Hellman in 1976 ([4],[5]). In 1977, Rivest, Shamir and Adleman invented the famous RSA cryptosystem [6]. In the RSA case, for instance, the trapdoor function is $c=f(x)=x^e(mod\, n)$ where n=p*q, a multiplication of two large prime numbers. The trapdoor is the numbers p and q, known only to the receiver. Knowing them, enables inverting the function by the receiver, that therefore for any C can find x.

Practical usage of the RSA and other public-key crypto systems are utilizing the principle of *Probabilistic Cryptography*, suggested first by Goldwasser and Micali [7] and OAEP that was proposed by Bellare and Rogaway [8] and subsequently standardized in PKCS #1 and RFC 2437.

Although several public key systems have been proposed, whose security relies on different computational problems, the most common ones are based on the factorization problem (e.g. RSA) or the discrete log problem (El Gamal, ECC).

Thus, the motivation for finding a one way trapdoor functions is clear. In this article, we will try to show a systematic way to compose a one way trapdoor function by utilizing the properties of ***VLSI logic synthesis algorithms***.

2 VLSI Logic Synthesis

VLSI Logic Synthesis is a widely used process that is a necessary phase in the design and manufacturing of VLSI devices, and FPGA images. During the *Logic Synthesis* phase, a logic design defined by boolean equations is converted to an electrical implementation that uses boolean gates, like NAND, NOR, OR, AND [1].

The Boolean Equation description of a given boolean function is usually called RTL (Register Transfer Level), and is written in a dedicated language (e.g. VHDL , Verilog are common in the industry).

The *Logic Synthesis* process is using a *Logic Synthesis Algorithm* that converts the RTL representation of a given function to a network of logic gates – the *Logic Network*, that represents the same function. The logic synthesis phase is implemented by a computer program called a synthesis tool. Synthesis tools are widely available in the commercial market from variety of vendors [2].

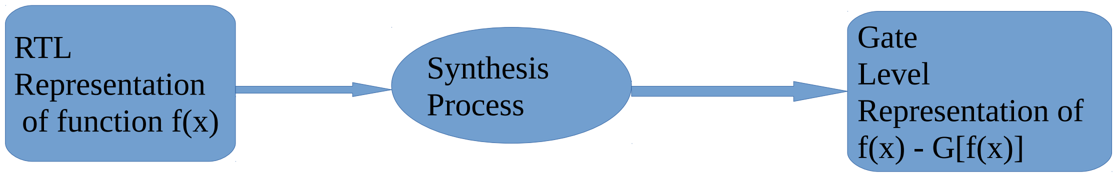
RTL Representation of function f(x)
Synthesis Process
Gate Level Representation of f(x) - G[f(x)]

Figure 1: The Synthesis process generates Logic Network representation of the function f(x)

We will use the notation of f(x) to signify a binary function with the vector x as its domain, and by the notation G[f(x)] to signify the Gate Level representation of f(x), that is the outcome of a synthesis algorithm operated on f(x), as depicted in figure 1.

2.1 Multi level logic minimization

A certain boolean function has an infinite number of gate level representations. Of those, there is a single one that is the smallest graph that represents f(x). The problem of finding this minimum is called *circuit minimization*. The general problem of circuit minimization is believed to be intractable, in the sense that there is no polynomial time algorithm that can minimize the logic network. However, there are variety of heuristics algorithm that find different local minimum of the network. As a result, multiple runs of the synthesis tool yields a different result for each run.

At the circuit minimization phase of the logic synthesis, constant values embedded in the function f(x) disappear, and the information associated with them is lost, and cannot be discovered from the network itself. The minimization phase of the logic synthesis eliminates all constant values and replaces gates associated with those values or exclude them from the network all together. As a result, it is impossible to gather any information about the original values of those constants [1].

3 Logic Synthesis Network as a One Way Function

In the this section, we will see why a synthesized network of logic gates may be refereed to as a ***One Way Function*** in the cryptographic sense.

In order to construct a *Public Key Encryption system*, one needs to find a one way function which is a bijection and that has a trapdoor. We will see how such a function can be constructed using Logic Synthesis algorithms and Gate level representation of a binary function. A ***logic network*** can be referred as a directed graph that its nodes are logical gates, mainly NAND, AND, NOT and OR, and its branches are binary functions of the inputs. A logic network can implement any desired logic function. Although in most cases the gates are electronic elements, here we are interested only in their behavioral characteristics.

3.1 Inversion of Logic network in special cases

in certain cases it is possible to invert a *logic network* easily: Those are the cases where the function f(x) represented by G[f(x)] is known and has a known inverse function $f^{-1}(x)$. In those cases, one can simply operate the synthesis algorithm to $f^{-1}(x)$ and thus constructs $G[f^{-1}(x)]$. If, for instance, we are given a logic network G[f(x)] while it is known that f(x) = x+4, knowing that $f^{-1}(x)=x-4$ we can easily construct $G[f^{-1}(x)]$ by direct application of the logic synthesis algorithm to $f^{-1}(x)=x-4$. Nevertheless, even in the cases where G[f(x)] is invertible, the inversion process is based on a priory knowledge of $f^{-1}(x)$.

3.2 Logic network inversion in the general case is not feasible

A logic network cannot be inverted based on the network graph itself, in general. Given a logic network G[f(x)], it is not known how to invert it in a polynomial time if $f^{-1}(x)$ is not known. That is, given a binary function f(x), and a gate level network G[f(x)] that represents f(x), there is no algorithm that can construct $G[f^{-1}(x)]$ from the network G[f(x)] itself in polynomial time, if $f^{-1}(x)$ is not known a priory. The only methodical algorithm known is the construction of a truth table of $f^{-1}(x)$, covering all its possible values, and operate the synthesis algorithm over the inverted truth table. However, since a truth table covers all possible inputs, a logic network G[f(x)] with, say, 128 binary inputs and 128 outputs would require a truth table table with a size of 2^{128} to be constructed. The complexity of this approach is exponential in the size of the input vector.

The main reason for the absence of an inversion algorithm that is based on the graph of the logic network itself is as follows: Except from the logic “NOT” gate, all basic binary logic gates are one way functions. From the output of any certain logic gate, it is impossible, in the general case, to reconstruct its inputs. Furthermore, for most of the logic gates, the inverse function is not existing at all, as they map more than one input to the same value. Therefore, it is impossible to construct an algorithm that will “opposite the direction” of the graph.

3.3 Preserving the “one wayness” properties of cryptographic primitives

In this section we will see why a logic network representation of one way function preserves the one way property. A cryptographic ***one way function*** (OWF), is, by definition, a function that is “easy” to evaluate but “hard” to invert. The terms “easy” and “hard” are used in order to express the fact that there is a complexity gap between the effort required for the evaluation of the function, to the effort required to evaluate its inverse. For example, if f(x) can be calculated in polynomial time but $f^{-1}(x)$ can be calculated only in exponential time, or there is not known way to evaluate $f^{-1}(x)$ in polynomial time, than f(x) is a one way function.

Lets say that we are given a one way function f(x). Since the inverse of the function $f^{-1}(x)$ cannot be evaluated at polynomial time, as explained in section 3.2, the logic network G[f(x)] that represents this function is also one way function in the sense that it cannot be inverted in polynomial time.

Example: Lets look at the cryptographic secure hash function SHA-256 [3]. This function is believed to be a one way function. Lets look at its gate level representation, G[SHA-256(x)] when the input x is a 512 bit vector, and the output is 256 bit vector. Clearly, G[SHA-256(x)] cannot be inverted in polynomial time based on the graph of the logic network. If it would possible to invert it, we could impose a successful preimage attack on SHA-256 as follows:

- Operate a logic synthesis algorithm on SHA-256, and construct the logic network G[SHA-256(x)]
- Invert the logic network in a polynomial time, getting the graph $G[f^{-1}(x)]$
- Operate the inverted logic network on any image of SHA-256(x), getting an input x' s.t. SHA-256(x') = SHA-256(x), thus executing a successful pre-image attack.

Since SHA-256 is believed to be pre image resistance, this cannot be done, and thus the one-wayness property preserved by the logic network G{SHA-256(x)].

4 Logic Synthesis of AES with a given key yields a one way Function

One of the most well known pseudo random permutations (PRP) is the AES-128 block cipher. The function C=AES-128(k,m) is invertible when the 128 bit key K is known. However, when the key is not known explicitly, the function C=AES(k,m) is indistinguishable from random function (as a PRP) and cannot be inverted in polynomial time, that is, its a one way function.

Consider the logic network that is generated by logic synthesis process that is operated on AES-128(k,m) when instead of using the key K as 128 bits input to the function, we use it as a constant random variable that is embedded in the logic network. The effect of the substitution of a random constant instead of the key K input is that AES-128 that is naturally a function of 256 bit (128 bit key and 128 bit message) becomes a 128 bit one way function (a function of the message m). The result, $C=G[AES(m)]$ has 128 bit input vector (m) and 128 bit output (the vector C). The key is embedded in the network and due to the non reversibility of $C=G[AES(m)]$ cannot be recovered but only by the one that operated the synthesis algorithm and choose the random key K. This is shown in Figure 2 below:

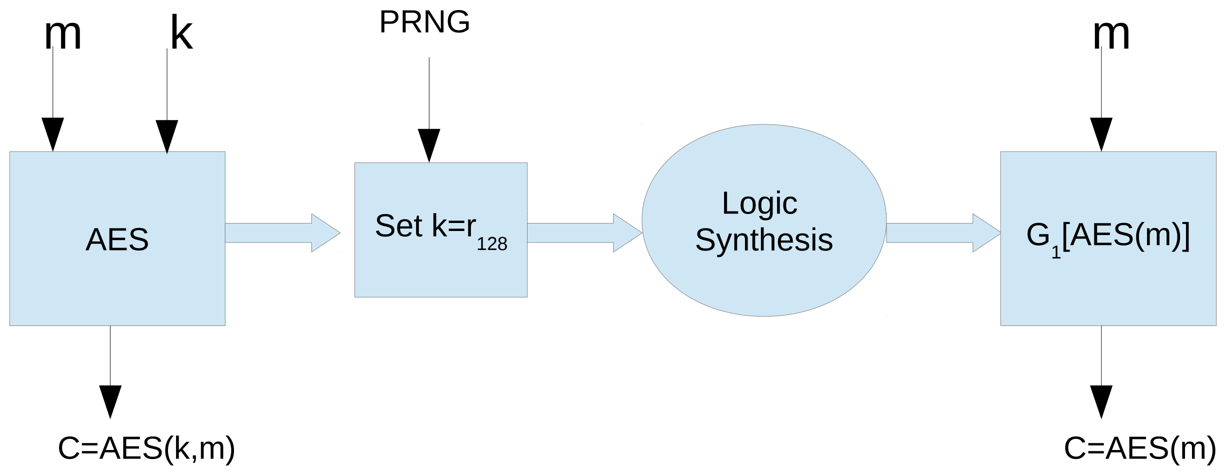

Figure 2: The generation of the logic network that represents AES(m)

The process of logic minimization, which is a part of the synthesis process, cancels any constant in the network and modifies the subsequent logic component so that any information about the constant K disappears. Furthermore, the logic minimization impose by the availability of a constant in the network spreads along the branches of the graph, cancels branches and nodes. And shrinks the network.

4.1 Improving G_1[AES(m)] immunity against invertability

Our target is to construct a trapdoor function by a logic synthesis of the AES cipher. For AES specifically, the first step is "*Add Round Key*", a logical operation that XOR all key bits with the message bits. The message is inverted where the key k bits equal "1", and remains un - changed where the bits of the key k are "0".
Since logic NOT is a reversible operation (As NOT is the only invertible logic function), inspecting the network G_1[AES(m)] defined in the previous section may yield some information about the bits of the key. The same statement is true also for the last step of AES, where the last round key is added as last step before the encrypted result is obtained.
In order to completely hide the bits of the key, K_r , we add two 128 bits random numbers, I_r and F_r (initial random and final random respectively). The resulting encryption and decryption equations get the following form:

$$\text{(1)} \quad C' = AES(K_r, (m + I_r)) + F_r$$
$$\text{(2)} \quad m = AES^{-1}(K_r, (C' + F_r)) + I_r$$

Where "+" stands for bit wise addition modulo 2. The improved synthesized network, G_2[AES(m)] is generated by the process that described in figure 3 below:

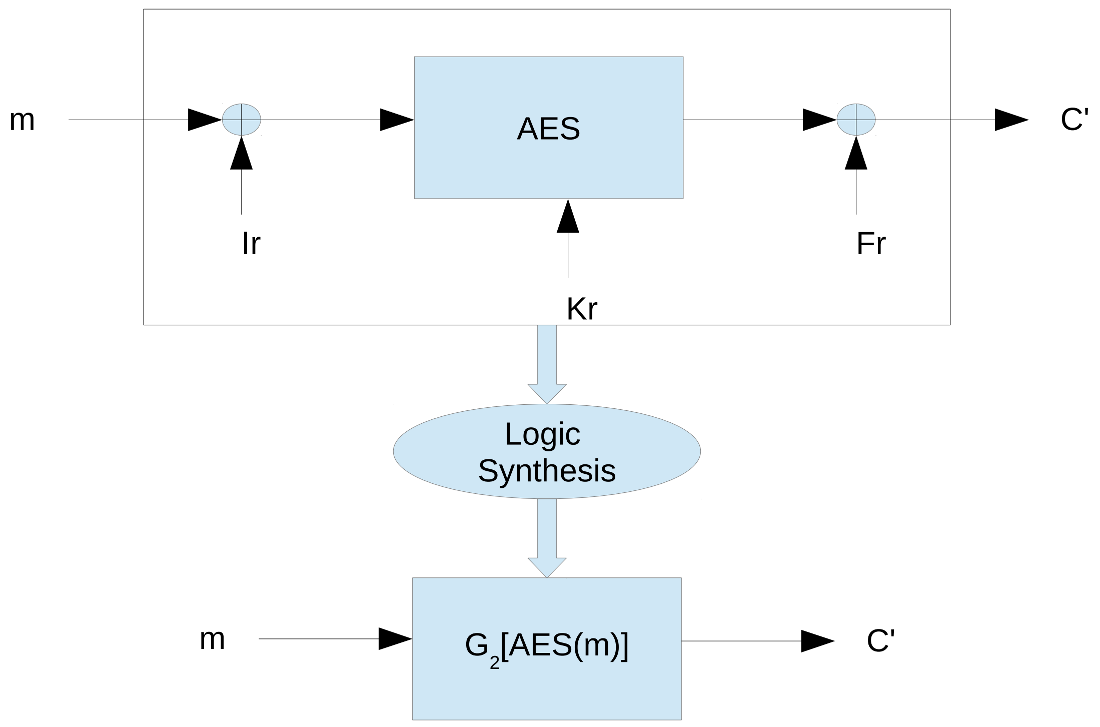

Figure 3: The generation of G_2[AES(m)]

The logic network G_2[AES(m)] represents equation (1) above. It has m, a 128 bit message as input, and it produces C', 128 bit number that is the encrypted value of m, as given by equation (1).

The decryption and recovery of m out of C' is done by calculating equation (2). In order to calculate equation (2), one need to know the random numbers Kr, I_r and F_r. Assuming that entity A generates G_2[AES(m)] by the process shown in figure 3, and publish the logic network G_2[AES(m)] , than entity B can use it to encrypt m.
Since entity B, or any other entity except from A does not know the secrets Kr, I_r and F_r , nobody except A can decrypt the message m. Note that since G_2[AES(m)] represents an AES cipher, knowing C' , a potential adversary cannot reproduce m, without knowing the key Kr.

An adversary may encrypt a lot of values of m using the public logic network G_2[AES(m)] . However, another property of the AES cipher is that knowing C' and m the adversary cannot recover the key Kr. Here the Adversary task is even harder, because he has also to recover I_r and F_r .

This leads to the following algorithm for the construction of a Public key system:

5 Public Key System Construction

5.1 Key Generation

1. Select three 128 bit random numbers K_r, I_r, F_r
2. Write the RTL description of $C' = AES(K_r, (m + I_r)) + F_r$ Where m is 128 bit binary input vector of the function, and C' is 128 binary output vector, and "+" stands for bit wise addition modulo 2
3. Synthesize $C' = AES(K_r, (m + I_r)) + F_r$ to get **G$_2$[AES(m)]** as described in Figure 3 above . **G$_2$[AES(m)]** is a logic network of 128 bit input (the binary vector m) and 128 bit output (the binary vector C'). The logic network **G$_2$[AES(m)]** has 128 signals as input and 128 signals as outputs
4. **G$_2$[AES(m)]** is the ***Public Key***
5. K_r, I_r, F_r is the ***private key***

5.2 Encryption

1. Select a message m, a 128 binary vector
2. Operate **G$_2$[AES(m)]** and generate C'
3. C' is the encrypted message

5.3 Decryption

1. Calculate the original message m by using equation (2) above: $m = AES^{-1}(K_r, (C' + F_r)) + I_r$ and using the standard AES algorithm

Note that decryption can be done only by the generator of the logic network **G$_2$[AES(m)]** , because he is the only one who has the values of K_r, I_r, F_r .

6 Security Discussion

A potential adversary may attack the above scheme by one of two ways:

- Either by trying to recover m from C'. This, if succeeded, breaks AES security because it is actually a successful Cipher Text attack on AES, as it detects m from C without knowing the key
- Trying to recover the key Kr from different values of m and C' that the adversary can encrypt as he knows **G$_2$[AES(m)].** This is actually a CPA on AES, as it tries to detect the key by applying the encryption to chosen plain text m and inspection of C'. So, relying on AES immunity to CPA, this attack cannot succeed.

Another way of detecting m or the key K is to try to reverse the logic network, but as explained in section 2 above, there is not known way how to do it in a polynomial time.

7 Conclusions

We presented a scheme for the construction of a public key encryption scheme that is based on Gate Level Synthesis of the AES block cipher. In this public key scheme, the logic network represents the public key and the random AES key and additional two random numbers compose the private key.

As a matter of fact, in addition to AES, any Pseudo Random Permutation (PRP) can be used by the above scheme in a similar manner, for example, DES, 3DES etc.

8 References

[1] *Synthesis and Optimization of Digital Circuits*, by Giovanni De Micheli, ISBN 0-07-016333-2.

[2] “Logic Synthesis Using Synopsys”, textbook, Pran Kurup, Taher Abbasi, Kluwer Academic Publication

[3] FIPS PUB 180-4 “Secure Hash Standard” March 2012 , NIST, http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

[4] W. Diffie and M. E. Hellman. Multiuser cryptographic techniques. In Proc. AFIPS 1976 National Computer Conference, pages 109–112, Montvale, N.J., 1976. AFIPS.

[5] W. Diffie and M. E. Hellman. New directions in cryptography. IEEE Trans. Inform. Theory, IT-22:644– 654, November 1976.

[6] Rivest, R.; A. Shamir; L. Adleman (1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". *Communications of the ACM* **February 1978**

[7] Shafi Goldwasser and Silvio Micali, Probabilistic Encryption, Special issue of Journal of Computer and Systems Sciences, Vol. 28, No. 2, pages 270-299, April 1984

[8]M. Bellare, P. Rogaway. *Optimal Asymmetric Encryption -- How to encrypt with RSA*. Extended abstract in Advances in Cryptology - Eurocrypt '94 Proceedings, Lecture Notes in Computer Science Vol. 950, A. De Santis ed, Springer-Verlag, 1995. full version (pdf)