EFFICIENT IMPLEMENTATION OF CRYPTOGRAPHICALLY USEFUL"LARGE" BOOLEAN FUNCTIONS

ABSTRACT

The art and science of keeping messages secure is cryptography and it is practitioned by cryptographers. Cryptography is used to solve problems. It solves problems involving secrecy, authentication and integrity. We present a low cost hardware architecture consisting of large Boolean functions and stream ciphers for implementing cryptography. The main idea of implementing cryptography using this technique centers around the generation of pseudorandom bits using Linear Feedback Shift Registers implemented through the pipeline concept. This technique is based on symmetric key algorithm. We have presented the reasons for implementing cryptography using symmetric key algorithm rather then going for public key algorithm. Using this technique we can achieve cryptography using a low cost hardware architecture which is very efficient for providing security to our data.

1. INTRODUCTION

Cryptography provides the necessary tools for accomplishing private and authenticated communication and for performing secure and authenticated transactions over the Internet as well as other networks. It is highly probable that every single bit of information flowing through our networks will have to be either encrypted or signed and authenticated in a few years from now.

Historically, four groups of people have used and contributed to the art of cryptography: The military, the diplomatic corps, diarists, and lovers. Of these, the military has had the most important role and has shaped the field. Within military organizations, the messages to be encrypted have traditionally been given to poorly paid

clerks for encryption and transmission. Until the advent of computers, one of the main constraints on cryptography had been the ability of the code clerk to perform the necessary transformations, often on a battlefield with little equipment. An additional constraint has been the difficulty in switching over quickly from one cryptographic method to another one, since this entails retraining a large number of people.

Modern cryptography is based on *key*, denoted by *K*. This key might be any one of a large number of values. The range of possible values of the key is called the *keyspace*. Both the encryption and decryption operations use this key, so the functions are given as:

$$E_k(M)=C$$

$$D_k(C)=M$$

where *M* is the message and *C* is ciphertext. A *message* is nothing but plaintext. The process of disguising a message in such a way as to hide its substance is *encryption.* An encrypted message is *ciphertext.* The purpose of turning cipher text back into plaintext is *decryption.*

1.1 Need for symmetric algorithms

There are two general types of key-based algorithms: *Symmetric* and *Public-key*. The number and length of messages are far greater with public-key algorithms than with symmetric key algorithms. The conclusion was that the symmetric algorithm was more efficient than public-key algorithm. The *RSA* system belonging to public-key algorithm ran about one-thousandth as fast as *DES (Data Encryption Signature)* and required keys about ten times as large. Although it had been obvious from the beginning that the use of public key systems could be limited to exchanging keys for symmetric cryptography, it was not immediately that this was necessary. Symmetric cryptography is best for encrypting data. It is orders of magnitude faster and is not susceptible to choose cipher text attacks as with public-key.

1.2 Symmetric Algorithm implementation using Stream Ciphers

Stream cipher cryptography is a classical method of secure information exchange. In this method, the message is considered to be a bit-stream. Encryption is performed by bit-wise *XOR*ing the message bit-stream with a pseudorandom bit-stream. This gives the cipher bit stream. Decryption is performed by bit-wise *XOR*ing the original pseudorandom bit stream with the cipher bit-stream. Let $(M_i)_{i>=0}$,$(K_i)_{i>=0}$, and $(C_i)_{i>=0}$, respectively, be the message ,pseudorandom, and cipher bit streams. The enciphering operation is given by

$$C_i=M_i \oplus K_i, \qquad i>=0$$

The bit stream C_i is transmitted. At the receiving end, deciphering is done in the following way

$$C_i \oplus K_i=M_i \oplus K_i \oplus K_i=M_i, \qquad i>=0$$

One of the popular models of hardware-based stream ciphers is shown in Fig. 1.

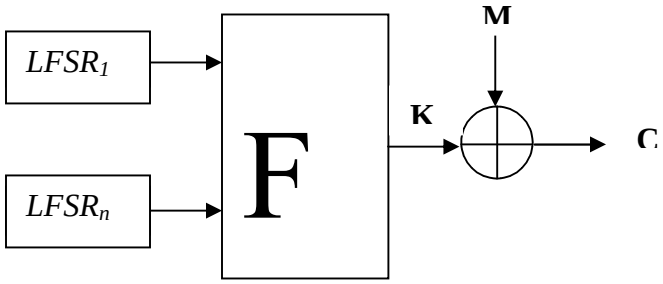


Fig-1. Stream Cipher System

In this model, t Linear Feedback Shift Registers *(LFSRs)* are combined using a Boolean function F to produce the pseudorandom bit-stream. At each clock cycle, each of the *n LFSR*s produces the pseudorandom bit-stream. These *n*-bits are combined by the Boolean function *F* to produce a pseudorandom bit. Thus, one pseudo random bit is produced at each clock cycle and, hence the rate of encryption is also one bit per clock cycle. The secret key of the system consists of the initial conditions of all the *LFSRs*. A function *F* of *n*-variables is built up from a function *h* of *k* (<*n*) variables. We describe an algorithm, which uses a subroutine for the function *h* and computes the output of *F* on an *n*-bit input.

2. IMPLEMENTATION

2.1 Algorithm

Let F be represented by $(h, S_1, \ldots, S_{n-k})$,where h is a function of k variables and let $t=n-k$. We have, $F_0=h$ and F_i is the function represented by $(h, S_1, \ldots, S_i)$. Then, $F_t=F$. A recursive algorithm to compute $F_t=F_{n-k}=F(X_n,\ldots,X_1)$ is as follows:

recCompute$(F_i(X_{i+k},\ldots,X_1)$

1. if($i=0$) return $h(X_k,\ldots,X_1)$;
2. if($\Psi i=Q$) then $\{X=X_{i+k};\}$
3. else$\{X=X_{i+k-1;}\, X_{i+k-1}=X_{i+k};\}$
4. if($X=0$) return *recCompute*$(F_{i-1}(X_{i+k-1},\ldots,X_1));$
5. else
6. if($\tau_i=c$) return $1\oplus$ *recCompute*$(F_{i-1}(X_{i+k-1},\ldots,X_1));$
7. if($\tau_i=r$) return *recCompute*$(F_{i-1}(1 \oplus X_{i+k-1},\ldots,1 \oplus X_1));$
8. if($\tau_i=rc$) return $1 \oplus$ *recCompute*$(F_{i-1}(1\oplus X_{i+k-1},\ldots,1\oplus X_1));$
9. end if

end

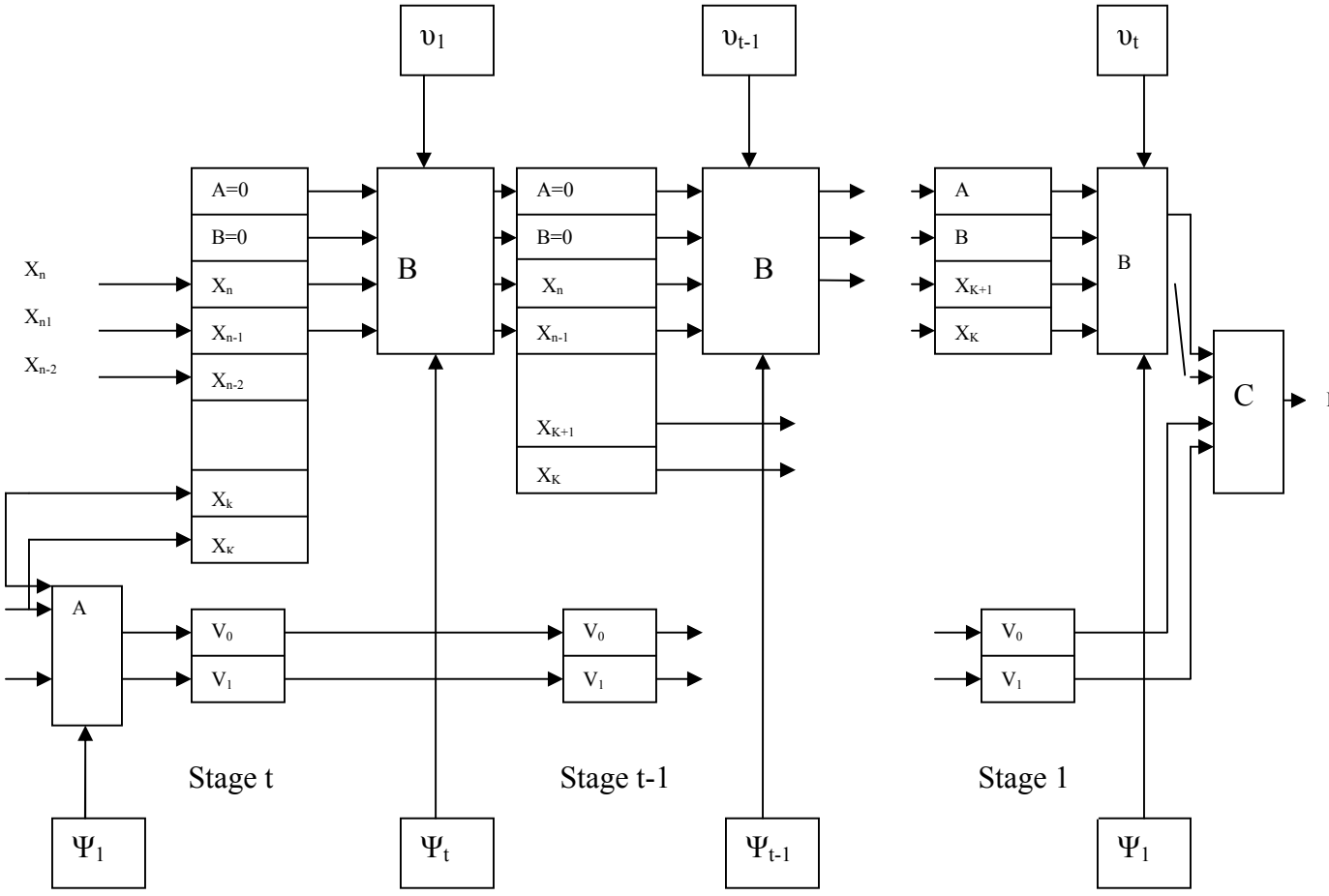
The *recCompute( )* takes t steps to compute a bit of output and stack depth is $O(t)$. Therefore having a large stack depth makes the algorithm inefficient to implement. Thus, we go for pipelined architecture implementation. The pipeline takes t cycles to be filled up, and after that it can handle an n-bit input at each clock cycle. For small t, there is no effective degradation in the throughput of the system.

2.1 Hardware Architecture

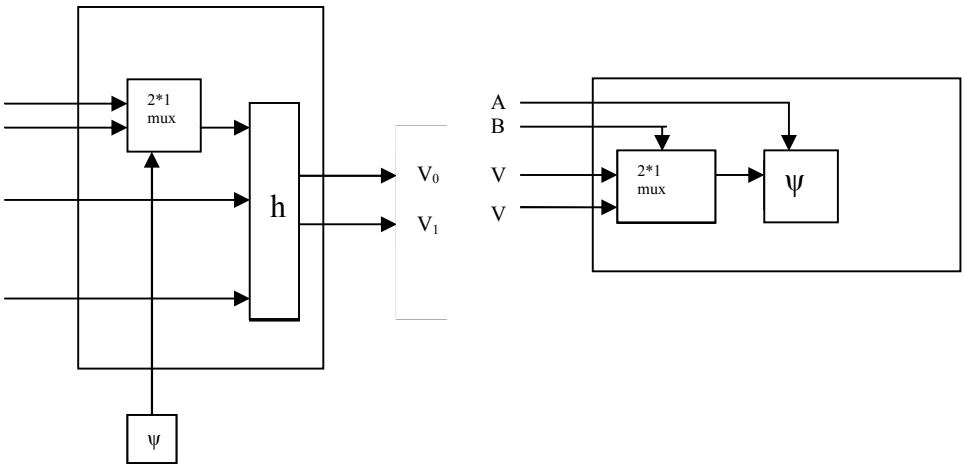
As mentioned above, a direct implementation of the algorithm requires t clock cycles to produce one bit of output. This will lead to unacceptable degradation in the performance of the system. Here, we show a low cost pipelined architecture can be developed to implement the circuit for F. The pipeline takes t clock cycles to fill up.

(a) Size of pipeline

The size of the pipeline in *Fig. 2* is the size of the look up table to compute *h* plus the additional gates and flip flops shown in *Fig.3*. We provide an estimate of the size.



Pipeline Architecture A : Initial circuit B: Intermediate circuit C: Final circuit.



Initial and final circuits (a) initial circuit (b) final circuit.

1. Each of the initial and final circuits requires a *2x1 MUX*. Additionally, the final circuit requires an *XOR* gate and the initial circuit requires a flip-flop to store the value of ψ_1.
2. Each of the i $(1 \leq i \leq t)$ intermediate circuits require

(a) two *2x1 MUX* circuits;

(b) two *AND* gates; one *OR* gate; one *NOT* gate; three *XOR* gates;

(c) $i+1$ flip-flops to store the values of $X_{k+i},\ldots,X_k$ and two flip-flops to store the values of ψ_i and τ_i.

As the number of stages grows, the cost will be dominated by the area required to store the values of the different X_js.

(b)Key Synchronization

The Boolean function is part of the stream cipher system. The working of the system requires a secret key to be shared between the sender and the receiver. A particular key is used to generate a fixed number of bits, after which it is replaced by a new key. The sender and receiver know exactly the points at which the new key is to be used.

In our case, the secret key consists of all the initial conditions of all the *LFSRs*. The use of a pipeline seems to suggest that, when a new key is used, the pipeline has to be flushed.

The same system will be available to both the sender and the receiver. Once both sides start with a specific key, the first output comes after a delay of *t* clock cycles, i.e, starting from *(t+1)*th clock. Now, consider the case when the key of the system is changed. When the new key is loaded, the pipeline will still contain some data generated from the earlier key. The data coming from the new key will be operational only after *t* clock cycles from the time it is loaded.

This is the same situation for both the sender and receiver. Both the sender and the receiver load the new keys at the same time. During the time the pipeline gets filled up with data from the new key, the bits from the old key continue to be used. Hence, there is no additional requirement for synchronization in this setup. The operation proceeds without any break in the generation of the pseudorandom bits.

3. EXAMPLE

We describe a 24-variable function *F*, which is built from a 10-variable function *h*. One possible representation *F* is

(h,(Q,r),(R,rc),(R,c),(Q,rc),(R,r),(Q,c),(R,r),(R,rc),(Q,r),(Q,rc),(R,r),(Q,rc),(R,r),(R,rc)).

Implementation of *F* requires a 14-stage pipeline. If *h* is fixed i.e., implemented by a combinational circuit, there are 4^{14} possible functions *F* which can be implemented

by a 14-stage pipeline. We now compare the sizes of direct and pipelined implementation of *F*.

Direct Implementation Size: The size of direct implementation of F is $\equiv 2^{24}$ gates/flip-flops.

Pipelined Implementation Size: The pipelined implementation requires:

a) 2,048 flip flops to implement the look up table for *h*.

b) 30 *2x1 MUXers*, 99 gates and 148 flip-flops.

The total delay in the system is 14 clock cycles. A direct implementation of *F* is prohibitively expensive. On the other hand, the pipelined is not only feasible, but requires only moderate cost.

4. CONCLUSION

. We conclude that we designed low cost hardware architecture to implement large Boolean functions, which is used in encryption of messages. Using this technique we can achieve an cost-effective way of encrypting our messages from the intruders. This technique is basically useful where we want to encrypt our messages using low cost hardware architecture implementation. Thus, using this technique we can save our information from intruders

REFERENCES

1. Bruce Schneier,"*Applird Cryptography*", WSE,second edition.

2. P.Sarkar and S.Maitra, "*Construction of Nonlinear Boolean Functions with Important Cryptographic Properties", Proc. Advances in Cryptology-EUROCRYPT 2000*,pp.491-512, 2000.

3. Y.V.Tarannikov,"*On Resilient Boolean Functions with Maximum Possible Nonlinearity", Proc.INDOCRYPT 2000*,pp.19-30,2000.