# Design of Power Efficient Symmetric Cryptography Algorithm

Mithil P. Gharat
Department of Computer Engineering
Vidyalankar Institute of Technology
Mumbai,India

Prof. Dilip Motwani
Department of Computer Engineering
Vidyalankar Institute of Technology
Mumbai,India

***Abstract*— Use of battery operated mobile devices increasing rapidly, use of computationally intensive cryptography techniques are not much efficient on mobile devices. To overcome this problem we are proposing modifications in Diffie-Hellman key exchange by merging it with RSA algorithm. To avoid man in the middle attack we try to encrypt the public components of Diffie-Hellman key Exchange using RSA Cryptosystem so they won't be accessible for any eavesdropper freely .RSA algorithm is used in this system also used to provide platform for authentication for users connected through insecure channels using Digital Signatures. This will also make Algorithm more complex to break while keeping computational complexity as low as possible.**

***Keywords*— KI, IR, PKC, DDH, PSS.**

## I. INTRODUCTION

The encryption algorithm is the chain of calculations that determine what ways the input plain text will be transformed into the output cipher text. There are two types of encryption: symmetric key encryption and public (asymmetric) key encryption. Symmetric key and public key encryption are used, often in conjunction, to provide a variety of security functions for network and information security.

Encryption algorithms that use the same key for encrypting and for decrypting information are called symmetric-key algorithms. The symmetric key is also called a secret key because it is kept as a shared secret between the sender and receiver of information. Symmetric key encryption is much faster than public key encryption, often by 100 to 1,000 times. Because public key encryption places a much heavier computational load on computer processors than symmetric key encryption, symmetric key technology is generally used to provide secrecy for the bulk encryption and decryption of information.

Encryption algorithms that use different keys for encrypting and decrypting information are most often called public-key algorithms but are sometimes also called asymmetric key algorithms. Public key encryption requires the use of both a private key (a key that is known only to its owner) and a public key (a key that is available to and known to other entities on the network). A user's public key, for example, can be published in the directory so that it is accessible to other people in the organization. The two keys are different but complementary in function. Information that is encrypted with the public key can be decrypted only with the corresponding private key of the set.

Performance is the primary concern so we try to keep complexity and battery consumption to the minimum level. . When performance is the primary concern, programming in assembly language e may seem to be the obvious choice, since coding in assembly provides control over the processor. However, it is very easy to use the wrong mix of instructions and the performance is often worse than code generated by a good optimizing compiler.

We tried to analyse well known symmetric ciphers AES, DES and Blowfish.

In case Advanced Encryption Standard Algorithm (AES) [3]. Increasing the key size by 64 bits of AES leads to increase in energy consumption about 8% without any data transfer [4]. In case of AES it can be seen that higher key size leads to clear change in the battery and time consumption. It can be seen that going from 128-bit key to 192-bit causes increase in power and time consumption about 8% and to 256-bit key causes an increase of 16%.

Data Encryption Standard Algorithm (DES) has a relatively small 56-bit key which was becoming vulnerable to brute force attacks. In addition, the DES was designed primarily for hardware and is relatively slow when implemented in software [6]. While Triple-DES avoids the problem of a small key size, it is very slow even in software; is unsuitable for limited-resource platforms, and may be affected by potential security issues connected with the (today comparatively small) block size of 64bits. The main disadvantage of DES is it can be cracked within three days by DES cracker.

Blowfish also improved from 224M memory size, but became steady in 352M memory size. Researchers this is because Blowfish function needs much more memory to initialize sub-keys and S-boxes [10]. In all, the Blowfish encryption algorithm will run 521 times to generate all the sub keys - about 4KB of data is processed that's why it takes more time for large data. Blowfish have disadvantage in decryption process over other algorithms in terms of time consumption and serially in throughput.

We try to encrypt the public components because, The Diffie-Hellman key exchange is vulnerable to a man-in-the-middle attack. For Example an opponent Carol intercepts Alice's public value and sends her own public value to Bob. When Bob transmits his public value, Carol substitutes it with

her own and sends it to Alice. Carol and Alice thus agree on one shared key and Carol and Bob agree on another shared key. After this exchange, Carol simply decrypts any messages sent out by Alice or Bob, and then reads and possibly modifies them before re-encrypting with the appropriate key and transmitting them to the other party. This vulnerability is present because Diffie-Hellman key exchange does not authenticate the participants.

There are two publicly disclosed prime numbers known as generator (g) and modulus (n) are used in Diffie-Hellman key Exchange algorithm. If these two are exchanged among parties using RSA encryption, then to find such Diffie-Hellman Secret Key one has to break the RSA encryption. Thus, it will be immensely difficult for an eavesdropper to find the secret key, which can then be used by end users for encryption and decryption purposes. Of course, RSA will be employed for the User Authentication purpose as well. In this fashion the computational complexity does not increase for end users and at the same time data is also M*N times more secure; M, N are the complexities of solving DLP (Discrete Logarithm Problem) and Integer Factorization Problem respectively.

## II. ISSUES FOCUSED

There are four issues which are focused mainly to provide for better security

### *A. Secure Key Exchange*

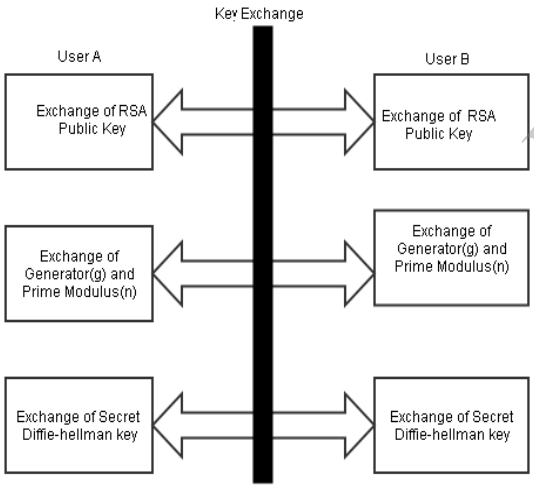


Figure 1. Key Exchange.

In our own approach we do not wish to send encrypted secret key along with encrypted data rather we try to generate same secret key at both ends using Diffie-Hellman key exchange policy. Moreover we exchange public components of this algorithm keeping them encrypted with RSA encryption decryption technique and therefore Secret Keys are far from being hacked.

### *B. User Authentication*

RSA algorithm is used in this system to provide platform for authentication for users connected through insecure channels using Digital Signatures. The RSA digital signature process also uses private keys to encrypt information to form digital signatures. For RSA digital signatures, only the public key can decrypt information encrypted by the corresponding private key of the set.

### *C. Degree of Security*

While exchanging Diffie-Hellman public components RSA is also used for user's authentication. Hence, all Man in the Middle attack can be brought to justice using Digital Signatures as evidence.

Using Brute Force attack, if somebody tries to reveal the secret key in computationally feasible time then he/she has to not only solve Integer Factorization problem in feasible time but also has to solve Discrete Logarithm problem at the same time.

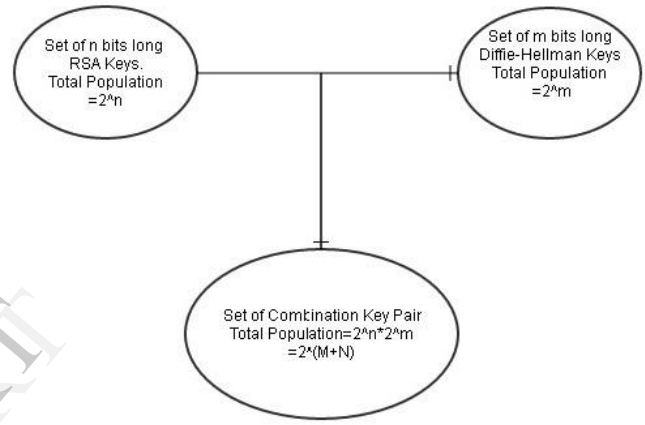


Figure 2. The set of combination of keys.

### *D. Computational Complexity*

Simply if we look at the computational complexities of both of RSA and Diffie-Hellman algorithm then we must realize that it takes more time with keys of larger size than that of smaller ones.

The algorithm will impose a computational complexity, which is equal to the multiplication of these two above mentioned algorithms because the public components of Diffie-Hellman are protected by RSA public key encryption

## III. ALGORITHM

There are three basic steps that constitute the whole process of Data Encryption, Data Transfer, Data Decryption and those steps are:

### *A. RSA Key Exchange*

Step 1: Generate RSA public components at user A and exchange those public components between user A & user B.
Step 2.Generate RSA private key at user A
Step 3: Generate RSA public components at user B and exchange those public components between user A & user B.
Step 4.Generate RSA private key at user B

(User A will possess public key of user B and its own private key. User B will possess public key of user A and its own private key.)

### *B. Secure Key Exchange*

Step1: Set p, g where p is a randomly chosen prime modulus and g is the generator for user A.

Step2: Set x, where x is a randomly chosen large number (This is the secret of user A).

Step3: Set Ka = g^x mod p.

Step4: User A encrypts g, p, Ka with the public key of the intended recipient of the message and sends it to User B.

Step5: User B receives encrypted g, p and Ka, sent by user A. User B then decrypts it and finds g, p and Ka.

Step6: Set y, a randomly chosen large number for user B.

Step7: Set $K_b$ = g^y mod p. Set DHKey = (Ka) ^y mod p

Step8: User B encrypts $K_b$ with the public key of User A.

Step9: User B then sends encrypted $K_b$ to user A.

Step10: User A receives encrypted $K_b$, sent by user B.

Step11: User A decrypts it and finds Kb.

Step12: Set DHKey = ($K_b$) ^x mod p (DHKey is the secret Diffie-Hellman key for user

(DHKey is the secret Diffie-Hellman key for user B).

Using TCP connection we can send and receive data and/or keys generated through this algorithm.

As TCP is well known, it is out of the scope of this part of the discussion to elaborate on how TCP can be used to exchange Keys and/or encrypted data

### *C. Data Exchange*

Step1: At first user data and/or application data (assume this user as user A) is read from the system and converted into its corresponding Byte- Code.

Step2: Now the Byte-Code is converted into its corresponding big integer form.

Step3: Convert DHKey into Byte code and Convert it into corresponding big integer form.

Step4. Divide the Data into Blocks of Key size.

Step5. Perform the Modulo 10 addition on each bit of data in block with DHKey until all blocks are processed this will be Encrypted message.

Step6: Transfer this message to User B.

Step7: User B performs Exact opposite operation to decrypt the data.

TABLE I. ADDITION MODULO 10

| + | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 |
| 2 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 |
| 3 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 |
| 4 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 |
| 5 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 |
| 6 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 |
| 7 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 8 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 9 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Addition Modulo 10

## IV. CONCLUSION

Using this algorithm we can encrypt and decrypt user and/or application data very easily and the simplicity of this algorithm is the soul of this algorithm though it creates combinational complexity for a eavesdropper to decrypt the cipher-text but for end users this algorithm never poses any complex functional activity to perform. Without knowing the Diffie-Hellman Key decryption of the cipher in this system is analyzed further more. The hacker has to try all sort of combination of the RSA and Diffie-Hellman key to find the exact combination for the specific transmission. Moreover, as we did not disclose the Public Components of Diffie-Hellman Part (i.e. generator and the prime modulus) the eavesdropper must have to break the RSA first to only find them and then have to counter to find Diffie-Hellman Secret key. If it is assumed that one 1024 bit RSA key to get broken through brute force attack on a highly sophisticated processor with great computational power, takes O (n) operations in an average case and for a Diffie-Hellman key (1024 bit) the same takes O (m) operations in average then for this algorithm it will take O (n*m) operations to break a set of keys. If m = n then we can say that to break a pair of keys using Brute-Force Attack the complexity will rise up to O(n^2).

## REFERENCES

1. Ruangchaijatupon, P. Krishnamurthy, "Encryption and Power Consumption in Wireless LANs-N,'' The Third IEEE Workshop on Wireless LANs - September 27-28, 2001- Newton, Massachusetts.
2. J. Daemen, and V. Rijmen, "Rijndael: The advanced encryption standard," Dr. Dobb's Journal, pp. 137-139, Mar. 2001.
3. R. Chandramouli, "Battery power-aware encryption," ACM Transactions on Information and System Security (TISSEC), vol. 9, no. 2, pp. 162-180, May 2006.
4. K. McKay, Trade-offs between Energy and Security in Wireless Networks Thesis, Worcester Polytechnic Institute, Apr. 2005.
5. A. Nadeem, "A performance comparison of data encryption algorithms," IEEE Information and Communication Technologies, pp. 84-89, 2006. 460
6. P. Ruangchaijatupon, and P. Krishnamurthy, "Encryption and power consumption in wireless LANs-N," The Third IEEE Workshop on Wireless LANs,pp. 148-152, Newton, Massachusetts, Sep. 27-28, 2001.
7. Bruce Schneier. The Blowfish Encryption Algorithm Retrieved October 25, 2008, http://www.schneier.com/blowfish.html

8. A. Nadeem, "A performance comparison of data encryption algorithms," IEEE Information and Communication Technologies, pp. 84-89, 2006.
9. V. Denzer and A. Ecker, Optimal Multipliers for Linear Congruential Pseudo-Random Number Generators with Prime Moduli. Behaviour & Info Tech 28 (1988), pp-803.
10. J. H. Loxton, David S. P. Khoo, G. J. Bird and J. Seberry, A Cubic RSA Code equivalent to factorization, J. Cryptology, 5 (1992), pp. 89-150.
11. J. Stern, Advances in Cryptology EUROCRYPT'99, vol-1592, Lecture Notes in Computer Science, (1999), pp. 223-238, Springer-Verlag.
12. N. Koblitz and A. J. Menezes, A Survey of Public-Key Cryptosystems, Mathematics of Computation, SIAM Review, 46 (2004), 599-634.
13. A. Nicolosi, M. Krohn, Y. Dodis, D Mazi`eres, Proactive Two-Party Signatures for User Authentication, NDSS, (2003).
14. M. Bellare, S. Duan and A. Palacio, "Key Insulation and Intrusion Resilience over a Public Channel", Lect. Notes in Com. Sc. Vol. 5473, M. Fischlin ed, Springer-Verlag, 2009.
15. M. Bellare, A. Boldyreva, A. Desai and D. Pointcheval, "Key-privacy in public-key Encryption", Adv. in Cryptology - Asiacrypt 2001 Proceedings, Lect. Notes in Com. Sc. Vol. 2248, C. Boyd ed, Springer-Verlag, 2001.
16. D. Boneh , G. Durfee , "Cryptanalysis of RSA with private key d less than N0.292 ", IEEE Tran. On Inf. Theo., Vol 46, No. 4, pp. 1339--1349, July 2000
17. D. Boneh and R. Venkatesan, "Breaking RSA may not be equivalent to factoring", In Proceedings Eurocrypt '98, Lect. Notes in Com. Sc.,Vol. 1233, Springer-Verlag, pp. 59-71, 1998.
18. R. Canetti, H. Krawczyk, "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels"., Eurocrypt, 2001. Long version available at eprint.iacr.org/2001/040.
19. M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication", Adv. in Cryptology - Crypto 96 Proceedings, Lect. Notes in Com. Sc.Vol. 1109, N. Koblitz ed, Springer-Verlag, 1996.
20. M. Abdalla, M. Bellare and P. Rogaway, "DHIES: An encryption scheme based on the Diffie-Hellman Problem", Lect. Notes in Com. Sc.Vol. 2020, D. Naccache ed, Springer- Verlag, 2001.