Increasing Symmetric Key Lifetime by Controlled Randomness*

Dimitrios N. Serpanos and Artemios G. Voyiatzis
Department of Electrical and Computer Engineering
University of Patras
GR-26504
Greece
{serpanos,bogart}@ee.upatras.gr

Abstract

Key replacement in cryptographic building blocks (algorithms and protocols) is a required operation in order to meet the security requirements. The common practice is to have only one cryptographic key valid for any given time moment and periodically change this key. We propose a scheme that allows multiple keys to be valid at any moment and describe a method that allows the receiver to identify the key used for each encryption with minimum effort, while not allowing an attacker to deduce this information. The proposed scheme introduces little computational performance overhead at the price of superior security characteristics and lower bandwidth utilization for control information.

1. Introduction

Advances in digital communication technologies allows transfer of information from geographically dispersed areas. For scale of economy, communication channels between end systems are implemented through public or by other means shared communication links. Secure communications is a requirement that has to be met, especially for critical applications, like transactions of economic impact or in an industrial factory, where espionage is a threat.

The science of cryptography contributes algorithms and protocols for ensuring data privacy, data origin authentication and data integrity. Key management schemes control the use of cryptographic keys for the underlying building blocks (algorithms and protocols), in such manner that the security requirements are met and the efficiency is maximized for the specific application.

Key management refers to the life cycle of cryptographic keys [3]. Key management incorporates the operations of key generation, key distribution, key storage, key replacement and exchange, key usage, and key destruction.

*This work was partially supported by the EU E-Next project FP6-506869

We focus on the problem of key replacement and exchange. Periodic changes of cryptographic keys is a necessary operation as to ensure (i) minimal exposure of plain data in case of key compromise and (ii) minimal collection of encrypted data under the same key, as to harden cryptanalytic attacks.

From the six management operations, key replacement and exchange can be the only one that requires exchange of control information over the communication channel. For this, it is an attractive target for an attacker, since it does not require physical access to end systems that store the keys.

Designers of secure systems rely on the fact that keys are periodically changed, in order to maintain the high level of security offered by the cryptographic primitives. Key replacement operations define the rekeying period for an algorithm (how often the key should be changed) and the behavior of the system during transition periods where both an old and a new key may be valid. This can happen for example in a loosely synchronized environment, where some clients can receive out-of-order packets from the servers.

In this paper, we explore a novel approach to the key replacement operation. The approach is to have multiple keys valid at any given time and allow the sender to choose from the set of valid keys the one to encrypt a packet. Our approach requires that the receiver has a means to deduce which key from the set of valid ones was used. This piece of information must be agreed between the sender and the receiver in a secure fashion.

The approach we propose has three desirable properties for protecting against cryptanalytic attacks: (i) the attacker must find which packets are encrypted under the same key, in order to then mount a specific cryptanalytic attack, (ii) consecutive packets that may carry redundant information are encrypted under a different key, and (iii) the disclosure of one key does not reveal the contents of a whole session; even if an attacker can know the time moments that the compromised key is used, he cannot resemble a whole session but only sketchy details of it.

These properties allow to extend a key's lifetime, both in terms of time of use and volume of data encrypted un-

der the same key. This extension of lifetime is desirable in many environments, since it achieves a higher level of security utilizing the same cryptographic primitives, it requires less frequent control messages to be exchanged, and it requires fewer management operations.

To the best of our knowledge, there has been no other proposal for allowing multiple cryptographic keys to be valid at any given time moment. The only exception are transient states between key changes, where a receiver due to synchronization skews may be required to try both the "old" and "new" key to derive the plain data.

The paper is organized as follows. Section 2 and 3 present an overview of current practices in key management and key replacement schemes. Section 4 introduces the idea of *controlled randomness* and Section 5 evaluates the proposed scheme. Section 6 concludes the paper.

2. Key management

In many real world problems, two parties need to exchange information in a secure manner over an insecure communication channel. Encryption algorithms, using a symmetric (secret) or an asymmetric (public) key are used to ensure such security. The *key management* problem refers to the process by which two or more parties use a common encryption/decryption key for their communication. *Key agreement* and *key transport* are the two main problems for a key management scheme. The former refers to the process by which the parties agree on a common key to use, while the latter refers to the process by which the key is transferred to all interested parties.

Key management schemes can be classified in three major categories:

- Public Key only (PK): a public key algorithm is used with a single public-secret key pair. The key is used to encrypt and decrypt the information for a long time. In an extension of this scheme, we denote it as PK-M, a new key is derived periodically, but the new key is transferred to the other communicating party using the current active key. PK-M does not offer forward secrecy and thus, it is not considered much safer than the simple PK.

- Secret Key only (SK): a secret key algorithm is used, with a single key shared by the two parties, as in PK. There also exist variations, such as the Secret Key - Master (SK-M), where a master secret key is known to the two parties; this key is used to derive ephemeral keys and exchanged data are encrypted with the ephemeral keys. Either the keys can be transferred along the channel (SK-ME), in analogy to PK-M, or feed a synchronized function in both sides that derive the same key periodically (SK-MR).

- Hybrid Key Model (HKM): a PK algorithm is used to transfer either a control signal for the SK-MR scheme above or to transfer the actual secret key to be used.

The HKM model is used in a wide range of applications because it combines the advantages of both PK and SK, while eliminating their disadvantages. In regard to SK family of key management schemes, we focus on block ciphers, because they constitute the base of SK systems. A stream cipher, which is the alternative to a block cipher, can be implemented either as the output of a block cipher or the output of a synchronized random number generator. Maintaining a synchronized random number generator is not a trivial issue and usually involves exchange of synchronization information.

All currently known key management schemes fall in one of these areas. One common characteristic of all these schemes is that at any given time instance at most one key is valid in the system. The only exception is at the time moments when the key changes, in cases where exact synchronization between the communicating parties is not feasible. In these cases, it is inevitable for a receiver to try to decrypt with both the old and the new key for a small time frame, in order to ensure that it remains synchronized with the sender on the correct key used.

This approach seems a wise one. Having one key valid at any given time simplifies management and contributes to efficiency as well. Furthermore, from a security point of view, it is quite challenging to design a scheme to transfer information on which key is used for each encryption, while not allowing an eavesdropper to access such information.

3. Key replacement and exchange schemes

The problem of key replacement and exchange refers to the frequency of key changes. Periodic key changes are necessary, since plain data tend to have redundant information. Redundancy is a means to attack a cryptosystem. Furthermore, key updates are necessary in order to limit the effects of a key compromise.

Hybrid key management schemes (HKM) are typically used for networked environments with many participants and point-to-point communications. A public-key scheme is used to periodically transmit a new symmetric key through the control channel. The symmetric key is used for subsequent encryptions and decryptions of the data channel communications.

In typical resource-limited environments, the public-key scheme is too costly to implement. For this, a symmetric-key algorithm is used in the control channel, using a *master*, pre-shared key, leading to an SK-M scheme.

In both cases, symmetric key cryptography is used in the data channel; the keys are periodically changed in order to ensure a high level of security. These keys are called *ephemeral* or *session* keys, since they have a limited lifetime. A higher level of security is achieved because the amount of consecutive data encrypted under the same key is minimized and thus, the exposure of one ephemeral key cannot expose all keys used (or at least this should be the

case).

We identify two main issues in this process. The first one is that consecutive and possibly redundant pieces of information are encrypted under the same secret key. Key updates offer a layer of protection, by limiting the number of ciphertexts per each key. However, advances in cryptanalysis may lower the number of ciphertexts needed for deriving a secret key. Countermeasures for such advances must be incorporated in the key management scheme in order to refresh the key more often. This results in more control messages in the channel, lowering the available bandwidth for actual and useful data and increasing the resources needed for public key decryptions, in the case of HKM, or increasing the data exposed by master key encryptions, in the case of SK-M.

The second issue is that a public key algorithm outputs typically 1024 or 2048 bits, while the secret key information encrypted using these algorithms is (again typically) 128 or 256 bits (64-bit keys are not considered secure currently). Thus, resources are wasted, in the form of bandwidth, since only 256 out of 1024 bits are useful, and in the form of computational resources and thus, energy consumption. Clearly, four or eight keys can be sent per public key encryption operation. Such a resource waste can be a significant overhead in resource-limited environments, such as embedded, battery-operated systems.

These two issues, drive our research: can we expose less information about which ciphertexts are encrypted under the same key? Can we better utilize the available bandwidth without sacrificing the security of the system?

4. Controlled randomness

Having at most one key valid at any time moment allows the attacker to focus his/her efforts on finding this specific key. Furthermore, having large consecutive pieces of information encrypted under the same key enables some forms of cryptanalysis, like known-plaintext or differential cryptanalysis. The same holds when implementing a control channel to transfer key management commands, like "new key is k_i" or "update key". An attacker can eavesdrop these messages, or at least detect their presence, which allows classification of the traffic to classes of data encrypted under the same key. If more than one keys are valid at any time instance, while no control information is used, then an attacker has harder work: now one can know that *a set* of keys is valid. One must classify the eavesdropped ciphertexts according to the key used for each ciphertext, in order to mount a differential or known-plaintext cryptanalysis attack.

In the absence of control information, the receiver needs a method to quickly decide which of the valid keys has been used. If control information is transmitted to the receiver, these information can be utilized by the attacker also. Thus, it must be avoided, if possible. One simple solution to this problem would be to decrypt received packets with all possible keys. Clearly, this approach does not scale well with the number of valid keys. The cost of the system is proportional to the number of decryptions that must be performed per time unit. In the next paragraphs, we propose a scheme that can identify efficiently the right key to use for each decryption, without utilizing a control channel.

We introduce the concept of *controlled randomness* for key replacement and exchange. The driving idea behind this concept is to allow multiple keys to be valid at any given time. The sender can choose for every packet or every few packets one key from the set and encrypt the data. In order to be both efficient and secure, such a scheme requires that the receiver must be able to easily identify which of the valid keys has been used for the encryption of a specific packet. To meet the security requirements, this information must not be available to an attacker that eavesdrops traffic exchanged between the sender and the receiver. In order to achieve this, we must address the following questions:

- What is the method that allows only the receiver to identify the correct key?
- How is the "control" channel protected from an eavesdropper?
- How this scheme hampers available bandwidth and application throughput?
- Is it efficient?

We provide the answers in the following subsections.

4.1. Key identification

A receiver can identify the correct key used for an encryption using at least two different methods, with respective advantages and disadvantages.

A first method is to have a synchronized random number generator with the sender. Both the sender and the receiver use this generator to produce a random integer between 1 and N, assuming that at most N different keys can be valid at any given moment. Since the two generators are synchronized, they can both know which key has been used for the encryption of each packet. However, an attacker eavesdropping the channel will not have this information available, since the information is never exchanged between the two ends. This method has the drawback that the two random number generators must remain continuously synchronized; in case of loss of synchronization, messages must be exchanged through a control channel to achieve resynchronization.

A second method is to utilize a Keyed Hash Function, commonly referenced as Message Authentication Code (MAC). The scheme operates as follows. The two entities, Alice and Bob, share a set of ℓ secret keys for a symmetric key cryptosystem and a set of ℓ secret keys for a message authentication code (MAC). Alice acts as the sending party and Bob as the receiving. For the moment, we do not identify how the keys are transferred between

Alice and Bob; this will be described in the next section. We use the notation $E_{alg}(m_j, k_i)$ to denote encryption of input data m_j with key k_i using the encryption algorithm alg. We denote concatenation of two strings of data with the symbol ||.

For each message that Alice sends, Alice:

1. draws a random number between 1 and ℓ, with uniform distribution,
2. sends $E_{alg}(m_j, k_i)||MAC(E_{alg}(m_j, k_i), h_i)$, i.e., a message m_j encrypted with the i-th encryption key and a MAC of the encrypted message, keyed with the i-th hashing key.

For each received message, Bob:

1. hashes the encrypted message with all possible hash keys, h_i, in order to find which one Alice used,
2. finds the corresponding position i,
3. decrypts using the i-th encryption key, k_i to derive message m_j.

This method of controlled randomness (utilization of MAC algorithms) has the advantage that it does not require the two parties to remain synchronized. Furthermore, from an implementation point of view, it requires only the sender to implement a random number generator.

In short, the controlled randomness scheme offers the following advantages:

- it uses a set of keys for each transmission instead of just one key,
- no key information or other control signals are transmitted over the channel during a session,
- it offers sender authentication through a MAC function,
- it offers data confidentiality though a secret-key encryption function.

The security of the proposed scheme is based on hash functions and, especially, message authentication code (MAC) algorithms, in order to avoid transferring key information through the communication channel.

Based on the advantages of the controlled randomness, the second proposed key management scheme has three major advantages:

- no need of synchronization is required between sender and receiver,
- no key information is transferred over the channel,
- a set of keys is not used sequentially (temporal keys); the attacker cannot know a priori which key is used at any given time frame and no signaling information is provided to help him detect whenever a new key is used.

In short, controlled randomness implements a method that uses concurrently all the ℓ different keys in one session (i.e., every key can be used at any time) and not linearly (use key k_1 for the first period, k_2 for the second, and so on). Figures 1 and 2 illustrate the linear use of keys in classical cryptosystems and the random but controlled use of keys as we propose respectively.

In our scheme, MAC functions are used to protect the secret keys rather than the message itself. The usage of a MAC algorithm enables us to avoid transmission of key information over the insecure communication channel.

5. Evaluation of the Controlled Randomness Scheme

5.1. Security

For the security evaluation, we will consider a session as a time frame. We also introduce Eve, which is an entity being able to eavesdrop the communication channel. In the session lifetime, we assume that at most ℓ different keys are used. We consider two known scenarios. In the first scenario, a secret-key algorithm is used, with a predefined key, k, which is used continuously to encrypt data. In the second scenario, a secret-key algorithm is used, with a predefined master key, k. At specific time moments, the two parties synchronize and derive ephemeral keys: k_1 for the first period, k_2 for the second, and so on up to k_ℓ for the ℓ-th and last period.

We compare the security offered by our scheme against these two scenarios. For all the three scenarios, we consider that the same secret key algorithm is used with a key of n bits size. For an otherwise secure secret key algorithm, the best known attack is not a brute force attack of complexity $O(2^n)$ but the birthday paradox attack of complexity $O(2^{n/2})$. This is the security offered by the first scenario. For the second scenario, it is known that the rekeying process increases the complexity of the attack to $O(2^{2n/3})$ for deriving the master key used [1].

In our scheme, we consider that the ℓ different keys are produced independently. Thus, an obvious improvement is that we achieve a linear increase in the attack complexity compared to the first scenario; the complexity becomes $O(\ell 2^{n/2})$.

In order to apply such an attack, Eve (the attacker) must know which key, or at least which index of the keys, has been used for each eavesdropped packet. Indeed, this would allow Eve to classify the packets in groups; in each group a single key, maybe unknown, has been used. However, this is infeasible, because Eve cannot derive the key or the key index, as we demonstrate below.

In our scheme, the sender uses a secure random number generator to decide which key to use in each period. Eve has two options in order to identify the key (or the key index) and perform the classification. The first approach is to attack the random number generator (RNG) and find its state and/or seed. We can safely assume that the RNG has a rather long period, that will render this attack unrealistic.

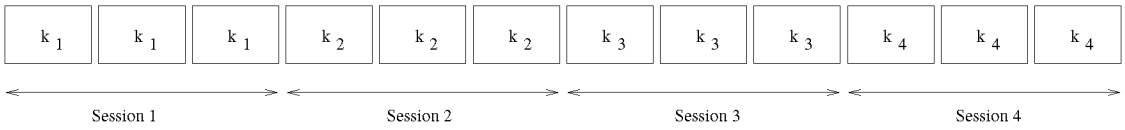

Figure 1. Linear use of keys during a session (classical cryptosystems)

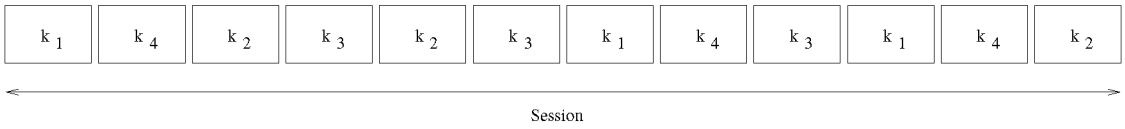

Figure 2. Concurrent use of keys during a session (proposed scheme)

For example, the ISAAC RNG has a state of $m = 16583$ or $m = 8295$ bits for 64-bit and 32-bit implementations, respectively [4]. In case Eve opts for this, she must perform $O(\ell 2^m)$ decryptions before being able to decide the state of the RNG. This is quite unrealistic, because in most cases $m >> n$ and thus, a brute force attack is preferable.

Eve's second approach would be to attack the MAC algorithm. Eve has as input the eavesdropped ciphertext $E(m_j, k_i)$ and can try all possible keys h_i in order to produce the eavesdropped hash $MAC(E_{alg}(m_j, k_i), h_i)$.

On average, $\min\{O(\ell 2^t, O(\ell 2^p)\}$ MAC operations must be performed, where p denotes the size of the MAC key and t the size of hashed message. Typically, $t = p$. After this work, Eve knows which ciphertexts are encrypted under the same key; she must then apply the best known attack for the secret key algorithm to derive the secret keys. So, the total complexity of an attack to derive all the secret keys would be $O(\ell(2^p + \ell 2^{n/2}))$.

An argument against the security of the scheme would be that the same keys are used quite often, compared to a traditional HKM scheme. It is important to note however that, it is not the secret keys that offer increased security; in our analysis we compare a worst-case scenario, where the keys in our scheme are used with the same frequency of those of the other scenarios. Recall that the motivation to change keys frequently is to not allow the attacker to collect enough information to mount an attack. Since the secret keys' usage is protected by the MAC, an attacker cannot derive the time instance when a specific key is used; to increase his efforts further, we can change each key's usage more frequently and thus, use more than ℓ and up to N different keys within a session.

5.2. Efficiency

The security analysis of the previous section indicates that our scheme would be preferable to deploy than a scheme with ephemeral secret keys, in terms of security. To further support this view, we emphasize that:

- no synchronization is needed between the server and the receiver to produce and use the same ephemeral key, in case they operate autonomously,
- there is no need for a "control" channel between the two parties to transfer new keys and/or other synchronization information. The hashed messages for each encrypted packet act as synchronization signals, but are incomprehensible by an eavesdropper.

A tempting approach to increase the efficiency of the scheme is to pass the index of the key or a hash of the key, instead of the MAC of the encrypted message. This would require less work by the receiving entity. However, as presented earlier, this lowers the security of the scheme dramatically, since an eavesdropper could derive which packets are encrypted with which key. Yet, the scheme is more secure than a *linear* encryption, in the sense that consecutive data are still encrypted with different keys. Such an approach could be desirable for heavily resource-constrained systems.

An argument against the efficiency of the scheme could be that the computation of ℓ MAC hashes poses a significant overhead in the system. Clearly, security does not come free. The scheme offers superior security than the currently known schemes. Furthermore, the implementations of MAC functions tend to be one order of magnitude faster than secret-key decryptions [2]. If we consider as c the work for a decryption, the work in our scheme can be as high as $c + \ell c/10$. We can decrease this number, without sacrificing security, by allowing multiple (say q) continuous packets to be encrypted by the same key. In this case, the work for the receiver becomes $qc + \ell c/10$, since the receiver needs to perform ℓ MAC operations every q packets. Careful selection of the parameters q and ℓ can ensure minimal overhead work; for example, a selection of $q = 10\ell$ results only to 1% overhead work.

5.3. Key updates

The proposed scheme does not describe a mechanism for key updates. Once the keys are setup in the system, they are used continuously. One may argue that we need to periodically update the keys for increasing the security. A public key scheme is a solution to this problem, resulting to a hybrid scheme. Traditional HK schemes use a public key algorithm, using a 1024 or 2048 bit key, and transfer a symmetric key as message, usually 128 or 256 bits. This result to underutilization of communication

bandwidth. Indeed, public key algorithms result to transmission of 1024 or 2048 bits, while containing only 128 or 256 bits of useful information, that is, the key for the symmetric algorithm.

We can group transmissions of symmetric keys to match the size of the public key. We can periodically update the keys by encrypting and sending $(k_i, h_i), \ldots, (k_j, h_j)$. This information can fit in one or more output of a public key encryption and thus result to higher utilization of consumed bandwidth. For example, consider the cost of transferring four new keys. In a traditional HK scheme, we need $4 \times 1024 = 4096$ bits for passing $4 \times 128 = 512$ bits, which is 12.5% bandwidth efficient. In our proposed scheme, we need to pass four symmetric keys of 128 bits each and four MAC keys of 128 bits each. Thus, we need $4 \times (128 + 128) = 1024$ bits, which is 100% bandwidth efficient. Furthermore, under our scenario, the receiver needs to perform only one costly public key decryption, instead of four, which results in less computation time and thus lower power consumption.

6. Conclusions

We introduced the concept of controlled randomness as a means to increase the security (lifetime) of ephemeral keys. This concept allows multiple keys to be valid at any given time, in contrast to schemes that define one key valid at each time instance. We proposed a scheme that utilizes this randomness and offers significant security gains, while adding little overheads.

We believe that the scheme is viable and offers an alternative method to implement secure communication protocols, while utilizing the same cryptographic primitives.

References

[1] M. Abdalla and M. Bellare. Increasing the lifetime of a key: A comperative analysis of the security of rekeying techniques. In T. Okamoto, editor, *Advances in Cryptology – Asiacrypt 2000 Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 546–559. Springer-Verlag, 2000.

[2] W. Dai. Speed comparison of popular crypto algorithms. `http://www.eskimo.com/~weidai/benchmarks.html`. Available, December 2, 2004.

[3] Jan C.A. Van Der Lubbe. *Basic Methods of Cryptography*. Cambridge University Press, 1998.

[4] R.J. Jenkins, Jr. ISAAC. In *Proceedings of the Third International Workshop on Fast Software Encryption*, volume 1039 of *Lecture Notes in Computer Science*, pages 41–49. Springer-Verlag, 1996.