Dynamic Encryption

Lars R. Knudsen

DTU Compute Technical University of Denmark
lrkn@dtu.dk

Received: 15 January 2015; Accepted: 2 February 2015;
Publication: 3 April 2015

Abstract

This paper considers the concept of dynamic encryption[1], which is about cryptosystems used for secrecy.

1 Introduction

Traditionally the participants in a secure communication scenario involving encryption need to agree on the particular algorithm used for the encryption and they need to establish a shared, secret key that is known only to the legitimate parties. Kerckhoffs' principle says that a cryptosystem should be secure even when attackers know everything about the system except for the value of the secret key (in a particular session). Here we introduce cryptosystems for which the receiver, Bob, does not know anything *a priori* about the system except for the value of the secret key, but without necessarily violating Kerckhoffs' principle and without sacrificing on security. The scenario is in more details: Alice and Bob exchange a secret key by some protocol. Alice (alone) decides on the cryptosystem to use, encrypts the cleartext with the agreed key and sends the cryptogram (and some additional information) to Bob. Although Bob does not know the cryptosystem used, the knowledge of the secret key is sufficient for him to decrypt the cryptogram and retrieve the message from Alice. The proposed method has many practical advantages and applies well to for example electronic mail systems, mobile conversation and

[1]Patent pending

Journal of Cyber Security, Vol. 3, 357–370.
doi: 10.13052/jcsm2245-1439.341

cloud storage. Moreover, the transmitting party in a conversation can change the cryptosystem as often as desired, e.g., for every new communication. This means that an attacker will have a harder time breaking the security, since he would first have to figure out how the encryption was effected, and thereafter he could try to break the encryption. This supports the strategy of "moving-target defense" that is becoming popular.

Encryption systems have existed for many years, several thousands of years if we are willing to believe the historicans. There are two main types of encryption, the symmetric-key encryption and the public-key encryption. The typical scenario of symmetric-key encryption is as follows, see also Figure 1. The sender, usually called Alice, and the receiver, usually called Bob, first agree on a key-exchange protocol and a particular encryption mechanism or encryption algorithm. Then they exchange a secret key *K*, such that after the execution of the protocol only Alice and Bob know the value of the secret key. Subsequently, Alice and Bob can exchange encrypted messages. In a public-key cryptosystem Alice and Bob each have a pair of keys, a private key and

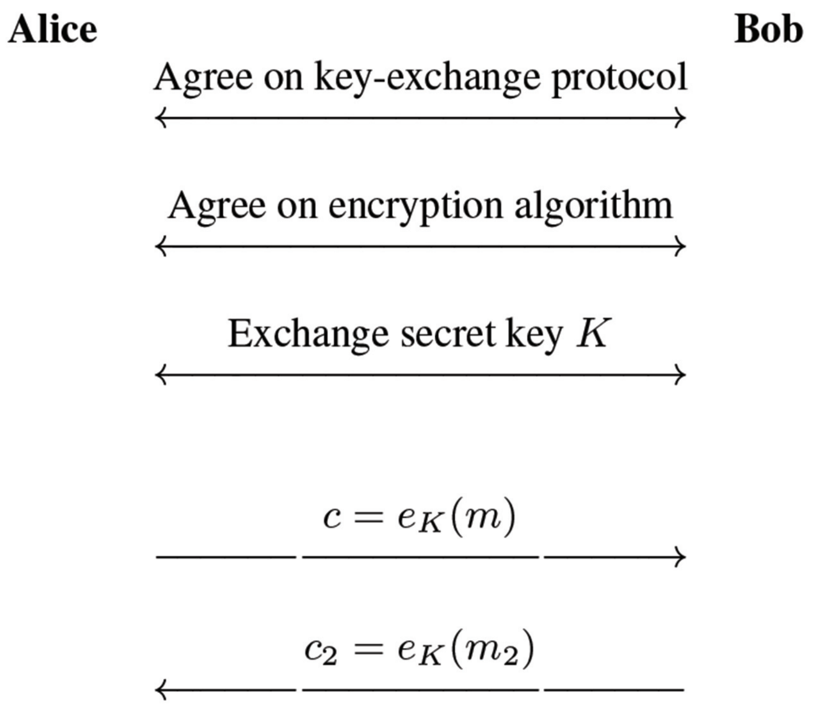

Figure 1 Typical scenario for symmetric-key encryption

a public key. The public key is made available for everyone and the private key is known only by one party. Then Alice uses Bob's public key to encrypt a message, and Bob can decrypt the message using his private key. Bob can send Alice a message by encrypting it with Alice's public key and Alice only can decrypt the message using her private key.

To evaluate the security of an encryption system it is customary to make at least two assumptions.

1. It is assumed that an attacker has access to the cryptograms sent between the sender and the receiver.
2. It is assumed that the attacker knows all details of the encryption process except for the value of the secret key.

The second principle is known as an interpretation of Kerckhoffs' principle [7, p. 225]. Kerckhoffs' principle is that the security of an encryption system should lie in the secrecy of the key and not in the secrecy of the system. Many examples from history (e.g., the World War II) show that this is a sound principle.

Today Kerckhoffs' principle is standard in cryptology-security evaluations. One advantage by using only public systems is that people will evaluate them and try to break them. A system which is kept secret can only be evaluated by the possible few people who knows about it.

2 New Approach

The new approach presented here can very shortly be described as follows: The receiver does not need to know any details of the encryption process except for the value of the secret key. Here the sender will choose the encryption system and encrypt in such a way that the receiver does not need to know how the encryption was done and still be able to decrypt correctly. And there are variants of this approach that do not have the disadvantage which comes from violating Kerckhoffs' principle.

In principle the new approaches can be applied to both symmetric-key and public-key encryption systems, but we shall concentrate on the former.

There are several ways of implementing the above idea, in this text we give two such examples.

2.1 Wrapped Encryption

It is assumed that the sender and receiver have agreed upon a secret key *K*. The sender does the following:

1. Choose (at random) a symmetric-key encryption system with encryption function E and decryption function D.
2. Construct an algorithm Ω which on input the secret key K and some ciphertext, implements D and returns the plaintext.
3. Compile Ω to the executable code X.
4. Send X to the receiver.
5. Encrypt the plaintext m using E and the key K and save the ciphertext c.
6. Send C to the receiver.

The receiver does the following:

1. Execute the code X.
2. Type in K and c on request.
3. Save plaintext m and delete X.

This algorithm is suitable for many different scenarios, including any data transmission e.g., email, voice transmission and cloud storage. There are applications where it may be advantageous to combine the transmission of X and c into one step. In this case, c can be incorporated into Ω. The compiled code X would then contain the ciphertext, which would not have to be transmitted separately. See Figure 2 for an example in the C programming language.

2.2 Encrypted Algorithm

It is assumed that the sender and receiver have agreed upon two secret keys $k_1$and k_2, and that they have agreed upon an encryption system ε. The sender does the following:

1. Choose (at random) a symmetric-key encryption system with encryption function E and decryption function D.
2. Encrypt D using ε with the shared key k_1 and transmit the result to Bob.
3. Encrypt the plaintext m using E and the key k_2 and transmit the ciphertext c.

The receiver does the following:

1. Decrypt the first part of the received ciphertext using k_1 to retrieve D.
2. Run D on the second part of the received ciphertext using k_2 to retrieve m.

In many applications, it would be acceptable that the encryption system ε is not very fast, since it is used only to encrypt a relatively short string. This would allow to choose a very strong cipher for ε, e.g., triple-AES.

```
#include ``stdio.h''

int ciphertext={``aeni92fj!~&k''};

void decrypt(int *ciphertext,int *key, int *plaintext)
{....

 plaintext = ....
}

main()
{
 printf(``Type in your key ``);
 scanf(&key);

 decrypt(ciphertext,key,plaintext);

 printf(``Plaintext is .... ``,plaintext);

 exit(0);
}
```

Figure 2 C-program example for data transmission

2.3 Discussion

Note that this approach does not conflict with using Kerckhoffs' principle. As an example, assume that there are *n* secure symmetric-key systems. Then the sender above can choose one of these systems at random for every message she sends. The receiver does not need to know which algorithm has been used to encrypt a particular plaintext. In §3 below we discuss some ways of constructing several symmetric-key systems from existing ones.

The main advantages and disadvantages of the approaches are the following:

Advantages

- *Adds another element of secrecy:*
  Since neither the receiver nor the attacker knows the cryptosystem which has been used for the encipherment, a successful attacker would need to first retrieve the encryption algorithm used, then try to break it. Moreover,

if the transmission between Alice and Bob is executable code, an attacker would have to decode the executable code into a higher level language in addition.

- *Adds another element of security:*
  Attacks on symmetric-key cryptosystem often require a huge amount of inputs and outputs from the encryption algorithm. If the encryption of many and/or long plaintexts is split between several secure encryption systems, one cryptosystem will be used to encrypt fewer plaintext blocks and the chance of a successful cryptanalytic attack will decrease. This supports the strategy of "moving-target defense" that is becoming popular [1, 4].
- *Adds efficiency to the system:*
  The receiver does not need to know the decryption algorithm, so this removes the need for the sender and receiver to first negotiate which encryption algorithm to use.

Disadvantages

For both approaches there will be some additional setup time in the system, and for the wrapped encryption approach:

- Transmission of executable code: users may be reluctant to execute program code received from other party. There are several ways to deal with this.
  - The sender can use a data integrity mechanism and send checkcode along with the other data. The receiver will check whether the received code is authentic before processing it. The integrity check should depend on the secret key. This is known as *symmetric-key authentication* and there are several international standards for this.
  - The code is executed in a protected programming environment. This would limit the risk in case of hostile code.

3 Customization

In this section methods we discuss how to construct private versions of existing encryption systems.

3.1 Variations of Existing Algorithms

Some encryption systems have been designed to allow for customizations, e.g., by choosing some of the components in the design according from a large set of primitives. As an example consider the *wide-trail strategy* behind the design of AES [8]. As for most symmetric encryption systems the AES can be split in some linear mappings and some nonlinear mappings. In the wide-trail strategy these two sets of mappings are constructed independently according to some predefined sets of constraints. For any components satisfying these constraints the result is a secure encryption system, where "secure" here is relative to the predefined constraints.

The estimated security levels of modern encryption systems are often calculated by assuming that the subkeys used in each iteration are independent. However, often the subkeys are computed from the shorter user-selected key in a so-called key-schedule. Further customizations of a system can be obtained by modifying the key-schedule.

3.2 Module Encryption

Also, one can use what we will call "module encryption". Most, if not all, symmetric-key encryption systems used in practice are so-called iterated ciphers. This means that the ciphertext is computed in a number of iterations, also called rounds, as a function of the plaintext and the secret key. In *module encryption* one constructs a number of iterated ciphers, say *s,* each with a relatively small number of rounds. Take the secret key *K* and generate from it a number of subkeys to be used in the small ciphers. Assume that the concatenation of *t* such small ciphers each dependent on a different subkey yield a strong cipher, resistant against all known attacks. In this way it is possible to select s^t variant ciphers.

3.3 Cascade Ciphering or Multiple Encryption

In a cascade of ciphers

$$\varepsilon_r \circ \ldots \circ \varepsilon_2 \circ \varepsilon_1$$

the encryption result of the first cipher ε_1, is encrypted again using the second cipher ε_2 and so. This is also called *multiple encryption* when the ciphers ε_i are similar.

It has been shown that for all attacks exploiting plaintext statistics a cascade of ciphers is provably at least as secure as the first cipher in the cascade [6]. Under a chosen plaintext attack the cascade of ciphers is at least as secure as any cipher in the cascade [2].

As an example, if ε_1 would be AES, any cascade of ciphers would be at least as secure as AES. Therefore one can implement dynamic encryption which is as secure as AES, yet the receiver and the attack do not know how the encryption was computed. Since AES is known, this implementation would not have the security risk one gets (traditionally) by violating Kerckhoffs' principle.

4 Variants of the AES

First we give a specification of the AES with the notation that we will use. The AES is an iterated cipher which runs in 10, 12, respectively 14 *rounds* depending on the size of the key of 128, 192, respectively 256 bits. These variants are named AES-128, AES-192, and AES-256.

The AES uses four main operations in a single round. In the following, let *r* denote the total number of rounds applied in encrypting a single block, thus for AES-128, $r = 10$. We use G_i, $1 \leq i \leq r$, to denote the round function which takes a 128-bit block as input and provides a 128-bit block as output. The ith round for $I \leq i \leq r - 1$ is defined as[2]

$$G_i = \mathrm{AddRoundKey}_i \circ \mathrm{MixColumns} \circ \mathrm{ShiftRows} \circ \mathrm{SubBytes}.$$

The final round of an AES encryption is special since MixColumns is omitted

$$G_r = \mathrm{AddRoundKey}_i \circ \mathrm{ShiftRows} \circ \mathrm{SubBytes}.$$

Before the first round, a pre-whitening key is used in a step $\mathrm{AddRoundKey}_0$, so the *r*-round encryption with master key *k* is denoted by

$$\mathrm{AES} - 128_k = G_r \circ \cdots \circ G_1 \circ \mathrm{AddRoundKey}_0.$$

Each of the four operations operate on a 128-bit block arranged in a 4×4 matrix over the finite field GF(2^8) defined via the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$. In this finite field, an element is represented by a single byte $a = (a_7 a_6 \cdots a_1 a_0)$, where $a_i \in \mathrm{GF}(2)$, which in turn represents the field element

$$a(x) = a_7 x^7 + a_6 x^6 + \cdots + a_1 x + a_0.$$

We use hexadecimal notation to write byte values. For example $a = 01$ represents the polynomial $a(x) = 1$ and $a = 02$ represents $a(x) = x$, and so on.

[2]The notation $g \ o \ f$ is function composition of f and g. The input to the composition is evaluated through f, the result is then fed to $g$and the output from $g$is the final result

In the following, we briefly describe the four operations used in an AES round. The text to be encrypted is 128 bits which is arranged as 16 bytes in a 4×4 matrix.

4.1 SubBytes

In the SubBytes operation, each of the 16 bytes in the state matrix are replaced by another value according to an 8-bit lookup table, an Sbox.

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \to \begin{pmatrix} S(a) & S(b) & S(c) & S(d) \\ S(e) & S(f) & S(g) & S(h) \\ S(i) & S(j) & S(k) & S(l) \\ S(m) & S(n) & S(o) & S(p) \end{pmatrix}.$$

For decryption one uses the inverse Sbox, which is easy to compute.

4.2 ShiftRows

In the ShiftRows step, the *i*th row of the state, $0 \le i \le 3$, is left rotated by i positions: ShiftRows:

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \to \begin{pmatrix} a & b & c & d \\ f & g & h & e \\ k & l & i & j \\ p & m & n & o \end{pmatrix}.$$

For decryption one uses right rotations instead of left rotations.

4.3 MixColumns

In this step, each of the four columns of the state matrix are multiplied from the right onto an invertible matrix M in the finite field. The matrix M is

$$M = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}.$$

For decryption one needs the inverse matrix which is

$$M^{-1} = \begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix}.$$

4.4 AddRoundKey$_i$

The r + 1 round keys, denoted $rk_0, \dots, rk_r$ are generated using the AES key schedule, cf. later. In this step, the 128-bit round key, rk_i, is added bitwise modulo two (using XOR) to the state.

The AddRoundKey operation is the same for encryption and decryption.

4.5 The AES Key Schedule

The round keys in AES are viewed as matrices with elements in the finite field GF(2^8). The first pre-whitening key rk_0 is the *n*-bit master key itself, so $rk_0 = k$. The key schedule varies slightly across the three AES variants. Here, we describe it for AES-128 and refer to the literature for the other two cases.

We consider the four columns of the two round keys as $rk_i = (rk_i^0||rk_i^1||rk_i^2||rk_i^3)$ and $rk_{i+1} = (rk_{i+1}^0||rk_{i+1}^1||rk_{i+1}^2||rk_{i+1}^3)$. To derive rk_{i+1} from $rk_{i,}$, $0 \leq i < r$, we do the following

1. Let $rk_{i+1}^j = rk_i^j$ for $j = 0, 1, 2, 3$,
2. Rotate rk_{i+1}^3such that the byte in the first row is moved to the bottom,
3. Substitute each byte in rk_{i+1}^3 by using the Sbox from the SubBytes operation,
4. Update the byte in the first row of rk_{i+1}^3 by adding 02^{i-1} from the finite field, and
5. Let $rk_{i+1}^j = rk_{i+1}^j \oplus rk_{i+1}^{j-1\, mod\, 4}\, for\, j = 0, 1, 2, 3.$

This procedure is repeated for i = 1,..., r to obtain the round keys rk_0, ..., $rk_{r.}$

4.6 Variants of the AES

There are several possible ways of making variants of the AES without destroying the ideas behind the design. The subfunctions in the encryption process are AddRoundKey, MixColumns, ShiftRows, and SubBytes.

AddRoundKey. The generation of round keys can be modified in many different ways without inviting to new cryptanalytical attack. However, it is not a trivial task to figure out how many secure ways there are of doing this.

MixColumns. This mapping is implemented by a circulant 4×4 matrix over GF(2^8). According to [3] there are around 2^{31} possible ways to choose such a matrix with properties similar to the one chosen in the original AES.

ShiftRows. There are 4! = 24 possible ways of choosing this mapping.

SubBytes. There are $2^8! \approx 2^{1684}$ possible bijection on eight bits, so there is plenty to choose from. In practice, one would probably prefer to use a (relatively) short key to pick one of these permutations, e.g., with a 128-bit "Sbox-key" we would pick one of 2^{128} permutations. Therefore we would only get a relative small subset of all permutations but we would still want to sample them uniformly.

To choose such an S-box one can use the Knuth shuffle which is simple and intutive [5]. The algorithm uses random numbers in the range [0, *i*] for varying values of *i,* which may not so easy to compute. An alternative is to use the initialization part of the stream cipher RC4, developed by Professor Ron Rivest from MIT [9].

5 A Dynamic Encryption Variant - Using AES

In this section one variation of dynamic encryption using the AES is presented.

We introduce a new cipher, RAES, which is similar to AES-128 except that the S-box used is different. RAES takes two keys each of 128 bits. One key is used just as the key in AES-128 is used, the other key is used to select the 8-bit S-box used in RAES. AES-128 has 10 rounds, cf., earlier, but RAES can be specified with less than 10 rounds.

Now we can define a dynamic encryption variant using AES.

DynAES

$$\mathrm{RAES}_{dk3,dk2}\mathrm{AES}{-}128_{dk1} \circ \mathrm{AddKey}\,[dk_0].$$

This is an AES encryption using dk_1 with a so-called prewhitening key dk_0, followed by an encryption using RAES, where dk_2 is used as the "AES-key" in RAES, and dk_3 is used to select the 8-bit S-box.

For decryption one goes in the reverse direction using the inverse Sbox in RAES, which is easy to compute.

5.1 GenerateSbox

There are many possible ways of generating an 8-bit S-box. We have chosen to use the key setup function of the stream cipher RC4.

The input to the function is an s-byte key $y = (y_0, \ldots, y_{s-1})$.

- let *T* be initialised such that *T(i)* = *i* for *i* = 0, ..., 255
- set *j* := 0
- for *i* := 0 to 255 do
  - j := j + $y_{i \ mod \ s}x$ + $K[i \text{ mod } s]$ mod 256.

– swap($T(i)$, $T(j)$)

The output is the table T.

The table T which is output from the key setup function can be as the Sbox in an AES variant. In the example above we used s = 16.

5.2 Security of the Variants

It is possible to prove that the above AES variant is at least as secure as the AES itself, but likely much more secure.

Theorem 1 *The cryptosystem DynAES where the keys* dk_0*,* dk_1*,* dk_2 *and* dk_3*are chosen independently at random is* **at least as secure** *as the AES-128.*

Also note that this proof is valid regardless of the number of rounds of the cipher RAES, also if RAES runs in zero rounds.

6 Conclusion

This paper has presented the approach of dynamic encryption. The sending party in a conversation can choose the encryption system at random from a set of many, secure cryptosystems, encrypt the cleartext and transmit the result together with some additional information. The receiver will be able to decrypt the received cryptogram on input the correct key. The receiver does not need to know how the encryption was performed, what is important for the receiver is to do the decryption and retrieve the message. The dynamic encryption approach has many applications.

References

[1] ACM/SIGSAC. First acm workshop on moving target defense (mtd 2014). http:// csis.gmu.edu/MTD2014, November 2014.

[2] Shimon Even and Oded Goldreich. On the power of cascade ciphers. In David Chaum, editor, CRYPTO, pages 43–50. Plenum Press, New York, 1983.

[3] Otokar Grosek and Pavol Zajac. Searching for a different aes-class mixcolumns operation. In *Proceedings of the 6th WSEAS International Conference on Applied Computer Science, Tenerife, Canary Islands, Spain,* pages 307–310, 2006.

[4] Homeland Security. Moving traget defense. http://www.dhs.gov/science-and-technology/csd-mtd. Retrieved November 26, 2014.

[5] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition*. Addison-Wesley, 1981.

[6] Ueli M. Maurer and James L. Massey. Cascade ciphers: The importance of being first. *J. Cryptology*, 6(1):55–61, 1993.

[7] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[8] National Institute of Standards and Technology. Advanced encryption standard. Federal Information Processing Standard (FIPS), Publication 197, U.S. Department of Commerce, Washington D. C., November 2001.

[9] R. L. Rivest, 1996. Attributed to Rivest in *Applied Cryptography* by B. Schneier, Wiley, 1996.