CS 161 Computer Security
Spring 2010 Paxson/Wagner
Discussion 6

March 09, 2010

1. One-Time Pads Recall how a one-time pad works. Alice shares a stream of random bits with Bob, and she encrypts a message of length n for Bob by XORing the next n bits of this stream with the message. Bob decrypts by XORing the ciphertext with the same n bits from the stream of random bits.

(a) Does this scheme work if we replace XOR with OR? How about with AND?

(b) Suppose you want to encrypt a message $M \in \{0,1,2\}$ using a shared random key $K \in \{0,1,2\}$. Suppose you do this by representing K and M using two bits (00, 01, or 10), and then XORing the two representations. Does this scheme have the same security guarantees of the one-time pad? Explain.

(c) Give an alternate encryption algorithm for carrying out the above task that does provide strong security guarantees. Note: You must not change the message space $\{0,1,2\}$ or the key space $\{0,1,2\}$. Instead, we want you to design an encryption algorithm $E(\cdot,\cdot)$ so that $E(K,M)$ is a secure encryption of M, when K and M are distributed as above.

Answer:

(a) No, it doesn't work with either OR or AND. First of all, correctness is broken; it is not true that $(m \vee k) \vee k = m$ for all choices of m and k (where m and k are each a single bit). Similarly, $(m \wedge k) \wedge k \neq m$ for all choices of m and k. This means that you can't actually decrypt an encrypted message.

Security is also broken. For OR, consider what an eavesdropper learns when she sees a 0 bit in the ciphertext. The only way this can happen is if both the key bit k and the message bit m are 0. For AND, when an eavesdropper sees a 1 bit in the ciphertext, she knows that both k and m are 1. Both OR and AND leak information.

(b) No, this scheme does not have the security guarantees of a one-time pad. The table below lists the resulting encrypted messages using this scheme. We can see that some outcomes exclude certain inputs. For example, given $E(K,M) = 11$ an attacker knows that the sent message M is not 0.

K	M	$E(K,M)$
00	00	00
01	00	01
10	00	10
00	01	01
01	01	00
10	01	11
00	10	10
01	10	11
10	10	00

(c) We wish to design a new encryption algorithm $E^*(\cdot,\cdot)$ that has the security guarantees of the one-time pad. We require that given $E^*(K,M)$, an attacker should get no information about M. This property is satisfied for any $E^*(K,M)$ that is uniform on $\{0,1,2\}$. One such algorithm is as follows:

$$E^*(K,M) = M + K \bmod 3.$$

The table below confirms that each outcome is equally likely.

K	M	$E^*(K,M)$
00	00	00
01	00	01
10	00	10
00	01	01
01	01	10
10	01	00
00	10	10
01	10	00
10	10	01

2. PRNGs and Stream Ciphers

(a) Pretend I have given you a pseudo-random number generator R. R is a function that takes a 128-bit seed s, an integer n, and an integer m, and outputs the n^{th} (inclusive) through m^{th} (exclusive) pseudo-random bits produced by the generator when it is seeded with seed s.

Use R to make a secure symmetric-key encryption scheme. That is, define the key generation algorithm, the encryption algorithm, and the decryption algorithm.

(b) Explain how using a block cipher in counter (CTR) mode or output feedback (OFB) mode is similar to the scenario described above.

Answer:

(a)
- **Key generation.** Generate a random 128-bit key.
- **Encryption.** Let j be the latest index we have used from our PRNG. We start with $j := 0$ and maintain the state of j for subsequent encryptions. Let L be the number of bits in message M. $E(K,M) = R(K, j, j+L) \oplus M$.
- **Decryption.** Define j and L as above. $D(K,C) = R(K, j, j+L) \oplus C$.

(b) CTR mode and OFB mode are both stream cipher modes. They use the key to generate a pseudo-random stream of bits. This key stream is then XORed with the message to form the ciphertext.

3. Properties of the Block Cipher Modes

The lecture notes describe four block cipher modes of operation: electronic code book (ECB), cipher block chaining (CBC), output feedback (OFB), and counter (CTR). Let $E_K(\cdot)$ and $D_K(\cdot)$ denote encryption and decryption of a single block under key K, respectively. Then given a message $M_1,\ldots,M_\ell$, each of those four modes defines a different way of computing the ciphertext blocks $C_1,\ldots,C_\ell$. Some modes also require a random initial vector IV to be selected before encrypting a message.

(a) Suppose $M_i = M_j$ for some $i, j \in \{1,\ldots,\ell\}$ where $i \neq j$. Which mode or modes (of the four) ensure that $C_i = C_j$ in this case? Is this good or bad?

(b) Encryption of messages with many blocks can be made more efficient if the individual blocks can be encrypted in parallel. This is possible whenever each ciphertext block C_i can be computed from only a fixed number of message blocks (that is, independent of ℓ) along with the IV (if there is one). Which mode or modes allow parallel encryption in this sense?

(c) Similarly, decryption can be parallelized if each message block M_i can be computed from only a fixed number of ciphertext blocks along with the IV (if there is one). Which mode or modes allow parallel decryption in this sense?

(d) Suppose a ciphertext block C_i is changed in transit. Then after decryption, some of the resulting message blocks will be corrupted. For each of the four modes, what are the indices of the message blocks that will be corrupted? How could you detect the fact that C_i has been changed in transit?

Answer:

(a) ECB. This is bad because it reveals information about parts of the message and completely breaks known plaintext security.

(b) ECB and CTR.

(c) ECB, CBC, and CTR.

(d) ECB i, CBC i and $i+1$, OFB i, CTR i. You could use a MAC to detect whether the ciphertext has been changed in transit.

4. Order of Authentication and Encryption Given a message M, an authentication function T_{k_1} (MAC or digital signature), and an encryption function E_{k_2} (symmetric or asymmetric), there are three choices of how to authenticate and encrypt messages. The ways you can do this are:

$$T_{k_1}(M), E_{k_2}(M) \tag{1}$$

$$E_{k_2}(M, T_{k_1}(M)) \tag{2}$$

$$T_{k_1}(E_{k_2}(M)), E_{k_2}(M) \tag{3}$$

(1) says send the encryption of the message and the authentication of the message separately. (2) says first authenticate the message, then encrypt the concatenation of the message and its authentication. Finally, (3) says to encrypt the message, then send the authentication of the encryption along with the encryption.

Which orders of authentication and encryption should one use? Are they all secure? If not, why? If yes, are there any trade offs?

Answer: (1) should not be considered secure. Remember that authentication does not provide confidentiality, only integrity. For example, some digital signature schemes (including RSA signatures) allow the verifier to recover the entire message based only on the signature. Thus, by sending $T_{k_1}(M)$ in the clear, one is potentially leaking information about M.

The choice between (2) and (3) is more difficult. There is a distinct trade off between the two. In the case of (3), there is a performance advantage in verifying authenticity of the message. In order to authenticate, it is only necessary to compute $T_{k_1}(E_{k_2}(M))$. However, in (2), it is necessary first to compute $D(E_{k_2}(M, T_{k_1}(M)))$ and then compute $T_{k_1}(M)$ to verify the MAC. Thus, it clearly requires more computation because in (3), the receiver never has to decrypt the message.

However, as a general principle, integrity is usually considered more important than confidentiality. Consider the damage Eve can do if she is able to modify the messages Alice sends to Bob compared to if she can just read the messages. In (2), it is harder to break integrity because the authentication is already encrypted. Basically, what is exposed to Eve is the encryption so that is what she can play with. In (3), the reverse is true. So in some sense, (2) has the advantage of "exposing" the less important secret to the attacker.

There is also a general principle that states "authenticate what you mean, not what you say." In the case of (3), we are not following the principle because we are authenticating the encryption, not the message itself. In (2), we guarantee that the message we decrypt is, in fact, what was intended for us to receive. This is relevant in a situation where the receiver might use the wrong key accidentally. Thus, they might authenticate that the encryption is correct, but use the wrong key to decrypt, getting gibberish and using it (there was a real bug in IPsec that did just this).

It should be noted, however, that there are theoretical results that show that (2) is technically less secure than (3). However, it turns out that this proof does not apply to the modes of encryption we generally use, such as CBC and CTR and generally is in a model of security not applicable to practical scenarios.

Basically, there is a trade off that you have to consider in selecting the order. Of note, famed security expert and cryptographer Bruce Schneier prefers (2). Just don't use (1).

5. **Diffie Hellman** Alice and Bob would like to use what they learned in CS 161 to communicate by email without letting anyone else read their letters. They do the following: Bob emails Alice his key g^x and Alice emails Bob her key g^y. They then use these to establish a secret key g^{xy}that they use to encrypt all their subsequent emails. Are they secure against eavesdroppers? Can anyone read their encrypted emails?

   Answer: Diffie Hellman is susceptible to MITM attacks. If Eve can intercept and inject her own messeges she can pretend to be Alice when talking to Bob and pretend to be Bob when talking to Alice.

   The general solution is to establish the keys in advance. If Alice and Bob both have published their keys, g^y and g^x, they don't need to trust the email channel when establishing their session key.