Cryptography Part 1 **Fall 2017**

Lecture 4: Authenticated Encryption Mode

Lecturer: Dan Boneh *Scribe: Lily Li*

So far we have discussed both confidentiality (via encryption) and integrity (via MACs), but we cannot really have one without the other. Consider sending an encrypted message. If we cannot ensure integrity, the attacker may change the cipher text so that the decrypted message had a different destination than was originally intended. Thus, when the TCP/IP stack intercepts and decrypts the message, it will send the message in the clear to some other destination.

4.1 Introduction to Authenticated Encryption

Definition 4.1 *An authenticated encryption system (E, D) is a cipher where $D : K \times M \times N \to C$ and $D : K \times C \times N \to M \cup \{\perp\}$ where N is an optional nonce and $\perp$ is a special symbol meaning that the message should be ignored. Here the system must be semantically secure under a CPA attack and have* ***cipher-text integrity****.*

Definition 4.2 ***Ciphertext Integrity*** *is defined in-terms of our usual security. The challenger choses a secret key $k \in K$. The adversary is free to send the challenger a sequence of message $m_1, ..., m_q \in M$ and receive $c_1 \leftarrow E(k, m_1), ..., c_q \leftarrow E(k, m_q)$ in return. Finally, the adversary sends cipher text c to the challenger. If $D(k, c) \neq \perp$, that is the cipher-text decrypts to a proper message, and $c \neq \{c_1, ..., c_q\}$ then $b = 1$ otherwise $b = 0$. As always we say that (E, D) has ciphertext integrity if for all* efficient *algorithm A:*

$$\mathsf{Adv}_{CI}[A, E] = \mathsf{Pr}[\textit{challenger outputs } 1].$$

Note: this definition does not directly protect against replay attacks, since a previously observed ciphertext can still be decrypted.

Definition 4.3 *Authenticated encryption defends against* ***chosen ciphertext attacks****. In this attack the adversary can mount both CPA and CCA (chosen ciphertext attack).*

Consider the security game definition. The challenger gets as input $b \in \{0, 1\}$ and choses a key $k \in K$. The adversary can ask CPA queries: pairs $(m_{i,0}, m_{i,1}) \in M$ such that $|m_{i,0}| = |m_{i,1}|$ getting $c_i = E(k, m_{i,b})$ in return, or CCA queries: ciphertext $c_i \in C$ such that $c_i \notin \{c_1, ..., c_{i-1}\}$ getting $m_i \leftarrow D(k, c_i)$ in return. The adversary needs to determine if they are in experiment 0 or 1. Specifically, $\mathbb{E}$ is CCA secure if for all efficient *algorithms A*

$$\mathsf{Adv}_{CCA}[A, \mathbb{E}] = |\mathsf{Pr}[EXP(0) = 1] - \mathsf{Pr}[EXP(1) = 1]|$$

is negligible.

Can you see why CBC with random IV is not CCA-secure?

Theorem 4.4 *Let (E, D) be a cipher that provides AE. Then (E, D) is CCA secure. In particular, for any q-query efficient adversary A there exists efficient adversaries B_1, B_2 such that*

$$\mathsf{Adv}_{CCA}[A, E] \leq 2q\mathsf{Adv}_{CI}[B_1, E] + \mathsf{Adv}_{CPA}[B_2, E].$$

Proof: (Actually this is only going to be a proof sketch.) Consider two separate games running simultaneously, one where $b = 0$ and the other where $b = 1$. We want to show that the chance that the adversary outputs one in both cases is identical. Since the challenger has cipher text integrity, the adversary cannot produce a valid ciphertext. Thus the challenger can output $\perp$ for all CCA queries (the challenger never gives any information away). The adversary knows that they will always get $\perp$ so will not issue CCA queries. What remains is an adversary which can only use CPA queries, and the advantage there is negligible. ■

4.2 Constructs from Ciphers and MACs

There are several ways to combine MAC with encryption. Not all of them provides authenticated encryption. The recommended ways is Encrypt-Then-MAC since encrypting first hides the data and computing a tag on the ciphertext locks it.

Alternatives include Encrypt-And-MAC which encrypts the data and appends a tag of the message (this is not safe since the tag could leak data about the message) and MAC-Then-Encrypt which first computes the tag then encrypts the message concatenated with the tag (this is not safe since there could be some bad interplay between the message and the tag).

4.3 TLS 1.2

The TLS Record protocol works as follows: each record has a header and a message body of length $< 16KB$. Both parties, browser and server, share a pair of keys $(k_{b\to s}, k_{s\to b})$. Each party also has two 64-bit counters used to recorded the number of messages passed during the interaction. This helps with replay attacks.

The encryption is done with CBC AES-128, and the MACing is done with HMAC-SHA1. The header consists of a type, version, and length of the message. The browser side takes as input $k_{b\to s}, data, ctr_{b\to s}$.

1. The browser first generates a pair of keys (k_{mac}, k_{enc}) from $k_{b\to s}$.
2. It then generates a tag using k_{mac} on the data: the incremented countered $ctr_{b\to s}$ concatenated with the header concatenated with the data.
3. Pad the header concatenated with the data concatenated with the tag to be the sized of an AES block.
4. CBC encrypt with k_{enc} and a new random IV.
5. Prepend the header.

4.3.1 Attacks on TLS (with CBC encryption)

Decryption with TLS takes as input $k_{b\to s}$, the record, and the counter $ctr_{b\to s}$.

1. CBC decrypt using k_{enc}.
2. Check to make sure that the pad has the right format: abort if invalid and throw a padding error.
3. Check the tag on $[++ ctr_{b\to s} \parallel header \parallel data]$: abort if invalid and throw a MAC error.

Since the attacker can distinguish the type of error that a chosen ciphertext produces, this encryption scheme is vulnerable to a padding oracle (even if the message returned is the same this scheme could still be vulnerable to a timing attack).

Suppose the attacker wanted to learn the first block $c[0]$ of the message. Throw away all cipher blocks $c[1], c[2], ...$ Produce a guess $g \in \{0, 1, ..., 255\}$ and give $c[0] \oplus g \oplus 0x01$ as the ciphertext. If the decryption returns with a MAC error then we know the pad $0x01$ is correct and thus the guess g is correct. To obtain the second to last byte, submit $c[0] \oplus g \oplus 0x0202$. Again, wait for a pad error. In this way a sixteen byte block can be broken with 16×255 queries.

The lesson you should take away from this is that it is much safer to Encrypt-then-MAC since the MAC is checked first and all invalid cipher texts are immediately discarded.

4.4 Attacking Non-Atomic Decryption

The concrete instance of this is in the SSH binary packet protocol. Each packet consists of a sequence number in the clear, the packet length, pad length payload and pad all encrypted, and a MAC tag calculated over the plaintext.

Decryption takes place as follows:

1. Decrypt packet length field.
2. Read as many packets as specified by the length field.
3. Decrypt remaining ciphertext blocks.
4. Check the MAC tag and send an error if invalid.

The problem is the non-atomicity of decryption. The time it takes to decrypt is directly proportional to the specified packet length — which is not verified. Thus the attacker can send any block as a supposed cipher text and obtain the first 32 bytes (packet length).

To prevent this include: do not encrypt but do MAC the length or adda a MAC of (seq-num, length) right after the length field.

4.5 Deriving Multiple Keys from a Single Key

Often in encryption protocols we need to derive a set of keys from a single source key. The proper way of doing this is to use a **key derivation function (KDF)**.

Let F to be a PRF with key space K and outputs in $\{0,1\}^n$. Suppose the source key SK is uniform in K. Define the KDF as:

$$\mathbf{KDF}(SK, CTX, L) = F(SK, (CTX \parallel 0)) \parallel F(SK, (CTX \parallel 1)) \parallel \cdots \parallel F(SK, (CTX \parallel L))$$

the **context**, CTX, is the unique to the application.

On the other hand, if the source key (SK) is not uniform then the PRF output will not look uniform either. This may leak information to an attacker.

4.5.1 Extract-then-Expand Paradigm

Here is how you are suppose to deal with non-uniform SKs:

1. Extract the pseudo-random key k from the source key SK using a **salt** — this is a fixed non-secret string chosen at *random*. The output should be indistinguishable from uniform.

2. Expand k by using it as a PRF key.

To implement this paradigm, extract by using $k \leftarrow HMAC(salt, SK)$. Then expand using HMAC as a PRF with key k.

4.5.2 Password-Based KDF (PBKDF)

Passwords are highly non-uniform so one should not use HKDF as described above. Otherwise the derived keys will be vulnerable to dictionary attacks. Thus PBKDF typically uses *salt* and a *slow has function* to make it more difficult for the attacker to mount such an attack.

The standard approach is PKCS#5 where the encryption is $H^{(C)}(pwd \parallel salt)$ or the hash function H iterated c (many) times. Then you can try to strike a balance between user experience (hash function still fast) and security (takes too long for the attacker to try the hash on all words of the dictionary.

4.6 Deterministic Encryption

Here we want the same message to always map to the same cipher text. You might recall that this type of encryption is not CPA secure (do you remember why), how ever this might still be useful in some cases e.g. encrypting database entries.

To solve this problem, we need to restrict our domain to only consider cases where there are unique messages. Using this restriction, we define the following:

Definition 4.5 *Let $\mathbb{E} = (E, D)$ be a cipher defined over (K, M, C). For $b = \{0, 1\}$ define $EXP(b)$, read experiment b, as:*

The challenger take b as an input and chooses a random key $k \in K$. For $i = 1, ..., q$, the adversary sends the challenger pairs of messages $m_{i,0}, m_{i,1} \in M$ where $|m_{i,0}| = |m_{i,1}|$. The challenger sends back $c_i \leftarrow E(k, m_{i,b})$. The adversary must output $b' \in \{0, 1\}$ which is their guess for b. Observe that the $m_{i,0}$ values are distinct and the $m_{i,1}$ values are distinct.

Note: an incorrect way to achieve deterministic CPA security is to use CBC with a fixed IV. The adversary would first send the pair of messages $0^n1^n, 0^n1^n$. The challenger would return the cipher text $c_1 \leftarrow [FIV, E(k, FIV), E(k, 1^n \oplus E(k, FIV))]$. Next the adversary would ask for the encryption of the message pairs 0^n and 1^n. Observe that this is a valid query since it does not repeat the previously asked messages but since the adversary can identify the encryption of 0^n, they have advantage 1.

4.6.1 Encryption Schemes for Deterministic CPA Security

4.6.1.1 Synthetic IV (SIV)

Let (E, D) be a CPA-secure encryption where $E(k, m; r) \to c$ such that r is a unit of randomness. Further suppose that $F : K \times M \to R$ is a secure PRF. Then

$$\mathsf{E}_{\mathsf{det}}((k_1, k_2), m) = E(k_2, m; F(k_1, m))$$

Essentially we are going to use $F(k_1, m)$ as r.

To see why $\mathsf{E}_{\mathsf{det}}$ is semantically secure under deterministic CPA note that since all the input messages are distinct $F(k_1, m)$ is a unique random string. Let $r = F(k_1, m)$. Since r is a random uniform string, E, by definition, must be CPA secure.

You should only use this encryption strategy for messages which are longer than one AES block. Why? (Hint: what makes the cipher text is longer than the message?)

Observe that SIV has a neat built-in check for integrity. The decryption algorithms gets the IV and the ciphertext. Decrypt the ciphertext to produce the message m' and generate the IV, IV' from m'. Check to make sure that $IV = IV'$. If so, then we know with high probability that the message is unaltered.

4.6.1.2 Using a PRP

Let (E, D) be a secure PRP with $E : K \times X \to X$. Then (E, D) is semantically secure under deterministic CPA. To see why this is the case observe that E is indistinguishable from a truly random permutation $f : X \to X$. The properties of f ensure that $f(m_{1,0}), ..., f(m_{q,0})$ is indistinguishable from $f(m_{1,1}), ..., f(m_{q,1})$.

Again this works mainly for short messages, but if you do want to use it for longer messages you could construct a wide block PRP using the EME construction (it is a bit unwieldy to explain here).

This system also does not have integrity, but you can add that by appending enough zeros to the end of your message before encrypting. If the decrypted message does not have these zeros as its least significant bits then the message has been tampered with. It is highly unlikely that a malformed cipher text will be able to preserve all the zeros.

4.6.2 Tweakable Encryption

You should think of this as: non-expanding encryption where the message space is the ciphertext space.

Since we do not have extra space to store randomness or check bits we can protect against at most deterministic CPA. It turns out (due to some theorem) that the only type of encryption possible here is a PRP however there are still some intricacies when it comes to applying the PRP.

If we use the same key k to encrypt ever sector of the data (here we are thinking primarily about storing data on disk), then the attacker will know which sectors are the same. Thus to scramble our message sectors we should use a different key for each sector. To generate all these keys it is a good idea to use a PRF as a pseudo-random key generator with a master key k. In particular, for sector t we will generate $k_t = PRF(k, t)$ then encrypt with $PRP(k_t, m[t])$.

An better way to do this is to use the master key k to construct many PRP's. Let $E, D : K \times T \times X \to X$ where t is the tweak. For every $t \in K$ and $k \leftarrow K$: $E(k, t, \cdot)$ is an invertible function on X indistinguishable from the random function. Now every sector will get its own PRP.

The security of a tweakable block cipher is also defined using the security game structure. Here the two experiments are sets of truly random permutations and the tweakable cipher E, the adversary gets to see encryptions of many tweak-message pairs and must guess which experiment they are in.

The trivial construction of a secure tweakable block cipher is as follows: let (E, D) be a secure PRP, with $E : K \times X \to X$. Then

$$E_{tweak}(k, t, x) = E(E(k, t), x).$$

However this procedure is quite slow since encrypting every sector requires two evaluations of E.

Consider instead the XTS tweakable block cipher construction: again we take (E, D) to be a secure PRP where $E : K \times \{0,1\}^n \to \{0,1\}^n$. Then compute the value $N \leftarrow E(k_2, t)$ and let

$$E_{tweak}((k_1, k_2), (t, i), x) = P(N, i) \oplus E(k_1, m \oplus P(N, i)).$$

where P is a fast padding function. The benefits of this construction is that only $n + 1$ evaluations of E are needed for n blocks of data.

Note: that encrypting the tweak t into the value N is necessary to ensure security. If we instead use the padding functions $P(t, i)$, the adversary can distinguish the distribution from random since they can easily check:

$$E_{tweak}((k_1, k_2), (t, 1), P(t, 1)) \oplus E_{tweak}((k_1, k_2), (t, 2), P(t, 2) = P(t, 1) \oplus P(t, 2).$$

Typically for disk encryption, each sector t is broken into blocks $b_1, ..., b_n$ and each block b_i is encrypted using tweak (t, i).

4.7 Format Preserving Encryption

Essentially the problem we wish to solve here is this: suppose we have structured data (e.g. credit card number) which we wish to encrypt. The cipher text is then passed through various intermediaries which all expect the data to follow the particular structure. How can we restrict the range of the ciphertext space to be that of the message space?

More formally: given $0 < s \leq 2^n$ how do we build a PRP on $\{0, ..., s-1\}$ from a secure PRF $F : K \times \{0,1\}^n \to \{0,1\}^n$ (e.g. AES for which $n = 128$)).

The construction is as follows:

1. Map from $\{0,1\}^n$ to $\{0,1\}^t$ where $t < n$ and $2^{t-1} < s \leq 2^t$ (recall that we want a PRP on $\{0, ..., s-1\}$). The way we are going to do this is to use the Luby-Rackoff construction with $F' : K \times \{0,1\}^{t/2} \to \{0,1\}^{t/2}$ then truncating F' to be of the appropriate length.
2. Map from $\{0,1\}^t$ to $\{0, ..., s-1\}$. We are given the PRP from the PRP (E, D) from the previous step and we want to build $(E', D') : K \times \{0, ..., s-1\} \to \{0, ..., s-1\}$. The construction is quite simple. Take input $x \in \{0, ..., s-1\}$ and set $y \leftarrow x$. Iterate: $y \leftarrow E(k, y)$ until $y \in \{0, ..., s-1\}$. Output y. Since $2^{t-1} < s \leq 2^t$, we expect two iterations.