CS 6160 Cryptology Lecture 11: CCA-Security & Message Authentication Codes (MACs)

Maria Francis

October 09, 2020

Defining CCA-security

- Chosen-ciphertext attack is even more powerful than eavesdropping and chosen-plaintext attacks.
- $\mathcal{A}$ has the ability to not only obtain encryptions of messages of its choice like CPA but also obtain decryptions of ciphertexts of its choice.
- Formally, that means $\mathcal{A}$ has access to a decryption oracle as well as an encryption oracle.

CCA indistinguishability experiment

$PrivK^{cca}_{\mathcal{A},\Pi}(1^n)$: We have the SKE $\Pi = (Gen, Enc, Dec)$, $\mathcal{A}$ and 1^n security parameter.

1. k is generated by running $Gen(1^n)$.
2. $\mathcal{A}$ has oracle access to $Enc_k(\cdot)$ and $Dec_k(\cdot)$.
3. It outputs a pair of messages m_0, m_1 of the same length.
4. A uniform bit $b \in \{0, 1\}$ is chosen and then a challenge ciphertext, $c \leftarrow Enc_k(m_b)$ is computed and given to $\mathcal{A}$.
5. $\mathcal{A}$ continues to have oracle access to $Enc_k(\cdot)$ and $Dec_k(\cdot)$. But it is not allowed to query on the challenge ciphertext.
6. Eventually, $\mathcal{A}$ outputs a bit $b^{'}$.
7. Output is 1 if $b^{'} = b$ and 0 otherwise. If output is 1 we say that $\mathcal{A}$ succeeds.

CCA-secure

Now that we have an indistinguishability experiment, we can have the security definition.

Definition
A SKE Π has indistinguishable encryptions under a chosen-ciphertext attack or is CCA-secure if for PPT adversaries $\mathcal{A}$ there is a negligible function negl such that:

$$Pr[PrivK^{cca}_{\mathcal{A},\Pi}(1^n) = 1] \leq \frac{1}{2} + \mathrm{negl}(n).$$

Also, if a scheme has indistinguishable encryptions under a chosen-ciphertext attack then it has indistinguishable multiple encryptions under a chosen-ciphertext attack.
Adversary has unlimited access to the decryption oracle except a request for the decryption of the challenge ciphertext itself.

CCA in the real-world

- The adversary may not get honest parties to decrypt arbitrary ciphertexts but it may be able to influence what gets decrypted.
- An adversary sends encrypted messages to the bank on the behalf of the user and see what the result is.
- For e.g what if the ciphertext corresponding to an ill-formed plaintext is sent and then the adversary may be able to deduce that from the bank's reaction.
- If encryption is part of the authentication protocol where one party sends a ciphertext to the other (using a PK), decrypts it (using SK) and returns the result to claim it is indeed him/her. The honest party being authenticated here is a decryption oracle!

CCA secure schemes so far?

- No, none of them were CCA-secure.
- Consider the CPA-secure encryption scheme we discussed in last lecture:

$$Enc_k(m) = \langle r, F_k(r) \oplus m \rangle.$$

- Let $\mathcal{A}$ run a CCA-indistinguishability exp with $m_0 = 0^n$ and $m_1 = 1^n$.
- On seeing $c = \langle r, s \rangle$, $\mathcal{A}$ flips the first bit of s and asks for decryption of $c^{'}$.
- $c \neq c^{'}$ and so this query is allowed.
- Decryption oracle returns either 10^{n-1} (in which case $b = 0$) or 01^{n-1} (in which case $b = 1$).

CCA secure schemes so far?

- The problem was we were able to manipulate the ciphertexts in a way that you still got a meaningful ciphertext with a relation to the original plaintext. We need to avoid this to avoid these attacks.
- I.e. CCA-security implies non-malleability.
- A non-malleable encryption scheme is s.t. if the adversary tries to modify a given ciphertext, the result is either an invalid ciphertext or one that decrypts to a plaintext having no relation to the original one.
- So far the CCA attack we discussed seems contrived, but you will see in public-key crypto they are more easy to visualize.
- For more practical CCA attacks in SKE, read Section 3.7.2 in Katz and Lindell textbook.
- We will also see how to build CCA-secure schemes in the coming lectures.

Chosen-ciphertext attacks (CCA)

- Some literature classifies them into two: Nonadaptive (CCA1 or Lunchtime attacks) and adaptive (CCA2).
- CCA1 or lunctime attack is when $\mathcal{A}$ has access to the decryption oracle only for a limited amount of time like for e.g. when the user of a computer is out for lunch.
- It is modelled as $\mathcal{A}$ chooses the ciphertexts to be decrypted without seeing any of the resulting plaintexts.
- An adaptive CCA or CCA2 is when $\mathcal{A}$ has unlimited access to the decryption oracle, before and after the challenge ciphertext, with the only condition that the challenge one cannot be queried.
- This is the one we looked at formally.

How to achieve CCA security?

- So at the end of the day, we have an active adversary Mallory that can send ciphertexts to Bob and get them decrypted.
- A layman view of CCA.
- What can Bob do?
- Use a CPA-secure encryption scheme but have Bob accept and decrypt only ciphertexts produced by Alice.
- Mallory can not create new ciphertexts that will be accepted by Bob.
- Building a CCA-secure Π now reduces to a problem of building a CPA-secure Π with message authentication.

Message Authentication

- Secure communication till now has been how to do secret communication over an open channel.
- Basically, we showed how encryption (private key) can be used to prevent an Eve or Mallory from learning anything about the messages over an unprotected channel.
- What if our security concerns are not related to secrecy?
- One other concern: how to guarantee integrity of our message? i.e. message authentication?
- Each honest party should be able to identify when a message it receives is indeed sent by the party that is claiming it has send it and not modified in between.

Message Authentication

- Consider the case when a user communicated with a bank and requests for a transfer of some money:
  - How does the bank know if it has indeed come from the user?
  - Even if it is from a legitimate user, how can we be sure that the message has not been tampered? As in the account number has not been changed for example?
- Standard error-correcting techniques wont work since they are for random errors and not for malicious ones which know what exactly to be changed.

Message Authentication

Another scenario where message integrity takes precedence: Web cookies.

- When you go to buy something from a shopping site, any state generated by a session for e.g: contents of your shopping cart is placed as a cookie with you, the client.
- It is sent to the server as part of each message you sent.
- If there are some items for which you get some special discount (user-specific pricing) then the server needs to make sure you have not modified the cookie to alter the prices of the product!
- Note: none of the details are secret.

Encryption Vs Message Authentication

- Encryption does not solve the problem of message authentication.
- Encryption using stream ciphers:
  - $c := G(k) \oplus m$ where G is a PRG.
  - Flipping any bit in c results in the same bit being flipped in the message.
  - Thus given a c , an encryption for m we can obtain a $c^{'}$ s.t. its decryption is the same as m with one bit flipped.
  - We can do the same attack for OTPs, so not even perfect secrecy guarantees integrity.

What about block ciphers?

- Same attack we described above works for OFB and CTR mode since they both generate a pseudorandom stream using block ciphers.
- This implies CPA-secure encryption schemes is not enough to prevent message tampering.
- What about CBC or ECB?
  - Note that ECB does not even provide CPA-security!
  - Both modes need inverting a PRF F and $F_k^{-1}(x)$ and $F_k^{-1}(x^{'})$ will be unrelated even if x and $x^{'}$ differ in only one bit.
  - But still there can be predictable changes in the plaintext.
  - In ECB, flipping a bit in the ith block of ciphertext changes only the ith block of plaintext.

What about block ciphers?

- In CBC mode, flipping jth bit of IV changes only the jth bit of m_1.
- All other plaintext blocks remain unchanged as $m_i = F_k^{-1}(c_i) \oplus c_{i-1}$ and c_i and c_{i-1} are not modfied.
- The first block of CBC (possibly header info) can be changed arbitrarily!

Defining MACs

- The communicating parties need to share a secret – in private key setting Alice and Bob share a secret key k generated by the *Gen* algorithm.
- A *MAC tag* or just *tag* t based on m and k is computed. It is computed by the *MAC* algorithm., $t \leftarrow MAC_k(m)$.
- It could be randomized and hence the $\leftarrow$.
- On receiver end, he runs the deterministic *Verify* algorithm on (m, t) to ensure that the given tag is valid, if valid output is a bit $b = 1$, else 0.
- Correctness:For all k generated by *Gen* and all messages m, $Verify_k(m, MAC_k(m)) = 1$.

Defining MACs

(m, t)

Alice

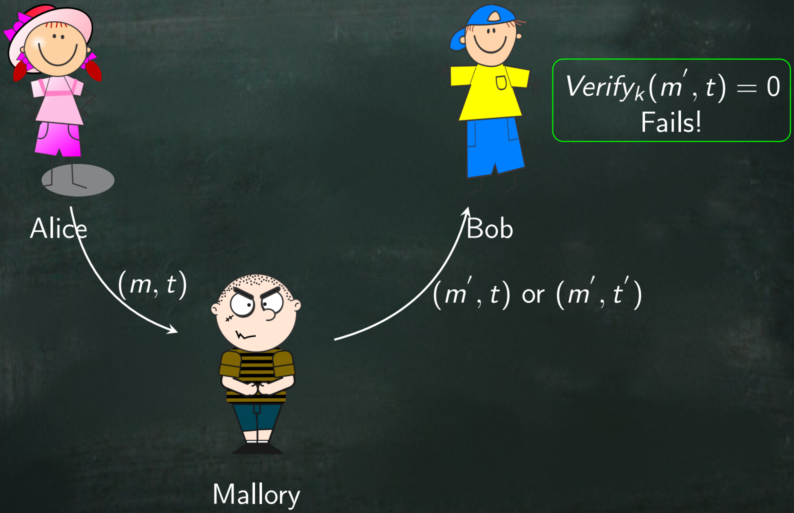
$t \leftarrow MAC_k(m)$

Bob

$Verify_k(m, t) = 1$

09/10/2020

Defining MACs



Security of MACs

- No efficient adversary should be able to generate a valid tag on a different message that was not previously sent.
- We need to formalize this idea, so we need to define a break of the scheme.
- Note that : an eavesdropping adversary can see all the messages sent by these parties along with their corresponding MAC tags.
- The adversary may also be able to influence the content of these messages, whether directly or indirectly. For e.g: the user changes the contents of the cookie stored on his computer.
- To formalize this we allow for $\mathcal{A}$ to request tags for any messages of its choice, i.e. access to an MAC oracle $MAC_k(\cdot)$.

Breaking MACs

- A break is when $\mathcal{A}$ produces (m, t)
  1. t is a valid tag for m.
  2. $\mathcal{A}$ did not request a MAC tag on m from the oracle.
- First case covers when honest parties are fooled into thinking that m came from a honest party.
- Second is a replay attack which is a serious attack *but not considered a break of a MAC*.I.e., $\mathcal{A}$ copies (m, t) sent previously by one of the legitimate parties.
- Before defining security we as usual give an experiment.

Message authentication experiment

$MAC - forge_{\mathcal{A},\Pi}(1^n)$:

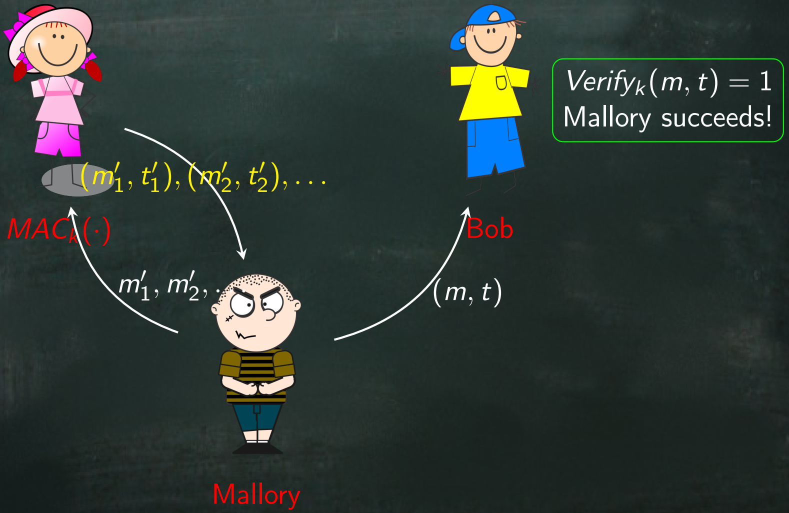
1. A key k is generated by $Gen(1^n)$.
2. $\mathcal{A}$ is given input 1^n and access to $MAC_k(\cdot)$.
3. After polynomial such accesses $\mathcal{A}$ outputs (m, t). Let $\mathcal{Q}$ denote the set of all queries that $\mathcal{A}$ makes to the oracle.
4. $\mathcal{A}$ succeeds with output 1 iff (1) $Verify_k(m, t) = 1$ and (2) $m \notin \mathcal{Q}$.

Definition

A message authentication code $\Pi = (Gen, MAC, Verify)$ is existentially unforgeable under an adaptive chosen-message attack, or just secure, if for PPT $\mathcal{A}$, there is a negligible function negl such that:

$$Pr[MAC - forge_{\mathcal{A},\Pi}(1^n) = 1] \leq \mathrm{negl}(n).$$

Message authentication experiment



Too strong a definition?

- $\mathcal{A}$ is allowed to request MAC tags for any messages of its choice.
- $\mathcal{A}$ succeeds if it can output a valid tag on any previously unauthenticated message.
- In real-life it would be meaningful messages that it should request MAC tags for and also succeed only for such a subset.
- What is meaningful? Too Application specific.
- Replay attacks are serious but MACs cannot work against them. Use sequence numbers or time-stamps.

09/10/2020

Strong MACs

- The specification has been that $\mathcal{A}$ cannot generate a valid tag on a new message that was never previously authenticated.
- But we can still have this scenario: an attacker might be able to generate a new tag on a previously authenticated message.
- More precisely, a *MAC* guarantees that if an attacker learns tags $t_1, \ldots,$ on $m_1, \ldots,$ then it will not be able to forge a valid tag t on any message $m \notin \{m_1, \ldots\}$.
- $\mathcal{A}$ may still forge a different valid tag ${t_1}'$ on m_1.
- $MAC-sforge$ takes care of that by considering $(m, t) \in \mathcal{Q}$, i.e. not messages but pairs of oracle queries and responses.
- A strong MAC is one in which the probability of $\mathcal{A}$ succeeding $MAC-sforge$ is negligible.

Canonical Verification and Strong MAC

- If MAC is deterministic, canonical verification is simply re-computing the tag and check for equality.
- $Verify_k(m, t)$ will first do $\bar{t} := MAC_k(m)$ and output 1 iff $t = \bar{t}$.
- If Π uses canonical verification then MAC is deterministic.
- A deterministic MAC can use canonical verification, but it doesn't have to.
- The following is easy to see (Assignment question!)

Theorem

Let $\Pi = (Gen, MAC, Verify)$ be a secure MAC that uses canonical verification ($\Rightarrow$ deterministic). Then Π is a strong MAC.