1 Construction of CCA-secure encryption

We now show how the MAC can be applied to obtain a CCA-secure encryption scheme. Recall that the attack we discussed (when applied to our CPA-secure encryption scheme) relied on the ability of the adversary to obtain forged ciphertexts, which then allowed her to recover the plaintext. It makes sense to apply a MAC to the ciphertext in order to prevent such forgeries.

For this application we will need to "upgrade" our definition of message unforgeability for MACs. Recall that a MAC (Tag, Ver) is (s, ε) secure against chosen message attack if any circuit $A^?$ of size s with access to a tagging oracle cannot produce a forgery with probability more than ε, where a forgery is a pair (M, T) such that $Ver(K, M, T) = 1$ and M is different from all the queries A submitted to its tagging oracle.

We will say a MAC is (s, ε) strongly secure against chosen message attack if under the same assumptions $A^?$ cannot even produce even the following weaker form of forgery: We say (M, T) is a *weak forgery* if (M, T) is different from all query-answer pairs from the interaction between $A^?$ and its tagging oracle. Here is the difference. If $A^?$ queries the oracle on M, obtains a tag T, and then outputs (M, T') for some $T' \neq T$ such that $Ver(K, M, T') = 1$, then (M, T') is a weak forgery but not a forgery. You can check that the MAC from last lecture is strongly secure against chosen message attack.

The scheme we construct will combine a CPA secure encryption scheme (Enc, Dec) with private key K_{CPA} and a MAC (Tag, Ver) with private key K_{MAC}. The resulting scheme $(Enc_{\mathsf{CCA}}, Dec_{\mathsf{CCA}})$ will have as its private key the pair $K_{\mathsf{CCA}} = (K_{\mathsf{CPA}}, K_{\mathsf{MAC}})$ and the following encryption and decryption algorithms:

$$Enc_{\mathsf{CCA}}(K_{\mathsf{CCA}}, M) = (C, Tag(K_{\mathsf{MAC}}, C)), \quad \text{where } C = Enc(K_{\mathsf{CPA}}, M)$$

$$Dec_{\mathsf{CCA}}(K_{\mathsf{CCA}}, (C, T)) = \begin{cases} Dec(K_{\mathsf{CPA}}, C), & \text{if } Ver(K_{\mathsf{MAC}}, C, T) = 1 \\ \texttt{error}, & \text{otherwise.} \end{cases}$$

Here is the intuition for this construction: For Eve to take advantage of the decryption oracle, she ought to query the oracle at some ciphertext that is different from its challenge. However, we now designed the decryption so that it returns the special message `error` unless the ciphertext C comes with a proper authentication tag T. So if Eve wants to make use of this oracle, she must query it at some pair $(C', Tag(K_{\mathsf{MAC}}, C'))$ where C' is different from C. But this requires Eve to produce a forged ciphertext. Since such a forgery is virtually impossible to produce (assuming the MAC is secure), the decryption oracle should reveal essentially no useful information to the adversary, because it will always return `error`. Then the adversary is only left with access to an encryption oracle, and so we have reduced our task to arguing CPA security, which will follow from the CPA security of (Enc, Dec).

One small complication is that we cannot actually assume the decryption oracle returns `error` most of the time, because Eve could run the decryption oracle on an output produced by the encryption oracle, in which case a valid tag will be produced. However, such queries are useless for Eve (since she already knows the answer), so without loss of generality, we shall be able to assume that Eve never makes such a query.

Theorem 1. *If (Enc, Dec) is (s, ε) CPA-secure and (Tag, Ver) is $(s, \varepsilon/s)$ strongly secure against chosen message attack, then $(Enc_{\mathsf{CCA}}, Dec_{\mathsf{CCA}})$ is $(\Omega(s/mt), 3\varepsilon)$ CCA-secure, where t is the circuit size of Enc and Tag and m is the message length.*

Proof. Let $s' = s/Cmt$ where C is a sufficiently large constant. For contradiction, suppose $(Enc_{\mathsf{CCA}}, Dec_{\mathsf{CCA}})$ is not CCA-secure. Then there is a circuit A_{CCA} of size s' and messages M, M' such that

$$\left|\Pr[A_{\mathsf{CCA}}^{Enc_{\mathsf{CCA}}, Dec^*_{\mathsf{CCA}}}(Enc_{\mathsf{CCA}}(K_{\mathsf{CCA}}, M)) = 1] - \Pr[A_{\mathsf{CCA}}^{Enc_{\mathsf{CCA}}, Dec^*_{\mathsf{CCA}}}(Enc_{\mathsf{CCA}}(K_{\mathsf{CCA}}, M')) = 1]\right| > 3\varepsilon. \tag{1}$$

Here we abuse notation a bit and use Dec^*_{CCA} for the decryption algorithm that is allowed to query every ciphertext except for the challenge provided as input to A_{CCA}.

We first replace A_{CCA} by an adversary that never queries its decryption oracle on an output of the encryption oracle. This can be done by maintaining a table of all the queries and answers that A_{CCA} makes to its encryption oracle and consulting this table before the decryption oracle is queried. The new adversary will have size $O(s'm)$ (with a careful implementation). Abusing notation we'll call this adversary A_{CCA} also.

We now want to use A_{CCA} to design an adversary A that breaks CPA encryption. We will make A simulate A_{CCA}, but every time A_{CCA} calls the decryption oracle, A will pretend that the decryption oracle returns `error`. More formally, $A^?$ is the following randomized oracle circuit:

> A^E: On input C, choose a random key K_{MAC} and simulate $A_{\mathsf{CPA}}^{E', D'}(C, Tag(K_{\mathsf{MAC}}, C))$, where E' is an oracle that on input M returns $(E(M), Tag(K_{\mathsf{MAC}}, E(M)))$, and D' always returns `error`.

Notice that $A^?$ makes at most $O(s'm)$ calls to the tagging algorithm and so it has circuit size $O(s'mt) \leq s$. By construction, $A^{Enc(K_{\mathsf{CPA}}, \cdot)}$ behaves exactly like $A_{\mathsf{CCA}}^{Enc_{\mathsf{CCA}}(K_{\mathsf{CCA}}, \cdot), Dec^*_{\mathsf{CCA}}(K_{\mathsf{CCA}}, \cdot)}$, except that the decryption oracle always returns `error`. (To simplify notation from now on we omit the private keys.) We want to argue that if A^{Enc} behaves very differently from $A_{\mathsf{CCA}}^{Enc_{\mathsf{CCA}}, Dec^*_{\mathsf{CCA}}}$ then A breaks the unforgeability of (Tag, Ver), and otherwise A breaks the CPA security of (Enc, Dec).

Formally, by (1), at least one of the following three conditions must hold:

$$\begin{aligned}
\left|\Pr[A_{\mathsf{CCA}}^{Enc_{\mathsf{CCA}}, Dec^*_{\mathsf{CCA}}}(Enc_{\mathsf{CCA}}(M)) = 1] - \Pr[A^{Enc}(Enc(M)) = 1]\right| &> \varepsilon \qquad \text{or} \\
\left|\Pr[A_{\mathsf{CCA}}^{Enc_{\mathsf{CCA}}, Dec^*_{\mathsf{CCA}}}(Enc_{\mathsf{CCA}}(M')) = 1] - \Pr[A^{Enc}(Enc(M')) = 1]\right| &> \varepsilon \qquad \text{or} \\
\left|\Pr[A^{Enc}(Enc(M')) = 1] - \Pr[A^{Enc}(Enc(M)) = 1]\right| &> \varepsilon.
\end{aligned}$$

The last inequality cannot occur because Enc is (s, ε) CPA-secure. So let us assume that the first inequality holds (the second one is symmetric). Since $A_{\mathsf{CCA}}^{Enc_{\mathsf{CCA}}, Dec^*_{\mathsf{CCA}}}(Enc_{\mathsf{CCA}}(M))$ and $A^{Enc}(Enc(M))$ are identically distributed conditioned on the decryption oracle never returning `error`, it follows that the probability of Dec^*_{CCA} returning something other than `error` in $A_{\mathsf{CCA}}^{Enc_{\mathsf{CCA}}, Dec^*_{\mathsf{CCA}}}(Enc_{\mathsf{CCA}}(M))$ is greater than ε.

By the union bound, there exists a query round i so that the decryption oracle returns `error` in rounds up to $i-1$ but not in round i is at least ε/s. If the decryption oracle does not return `error` in round i, then it must have been called on a query (C_i, T_i) such that $Ver(K_{\mathsf{MAC}}, C_i, T_i) = 1$. Now consider the following adversary A_{MAC} that implements a chosen message attack on (Tag, Ver):

> A^T_{MAC}: Choose a random key K_{CPA} and simulate $A^{E',D'}_{\mathsf{CCA}}(Enc(K_{\mathsf{CPA}}, M), T(Enc(K_{\mathsf{CPA}}, M))$, where E' is the oracle $(Enc(K_{\mathsf{CPA}}, \cdot), T(Enc(K_{\mathsf{CPA}}, \cdot)))$ and D' returns `error` for the first $i-1$ queries. Output the ith query (C_i, T_i) made to D'.

We claim that A^T_{MAC} is a chosen message attack on (Tag, Ver) that succeeds with probability at least ε/s. Notice that A_{MAC} only calls its tagging oracle when A_{CCA} calls its encryption oracle, in which case A_{CCA} stores the answer (C, T) in a lookup table. When A_{CCA} makes the ith query (C_i, T_i) to the decryption oracle, we know that (C_i, T_i) must be different from all such (C, T), for otherwise A_{CCA} would have just looked up the answer instead of querying the oracle. So with probability at least ε/s, A^T_{MAC} produces a weak forgery (C_i, T_i) for (Tag, Ver). Since $A^?_{\mathsf{MAC}}$ has size $O(s'mt) \leq s$, (Tag, Ver) cannot be $(s, \varepsilon/s)$ strongly secure against message attack, contradicting our assumption. □

2 How not to combine encryption and authentication

The design of the CCA-secure authentication scheme in the last section came about naturally as we attempted to resolve the malleability issue in the original, CPA-secure encryption scheme. The malleability of the original scheme allowed an active adversary to take the encryption of any message M and turn it into an encryption of another message M'. To prevent this problem, we realized that it is sufficient to authenticate the ciphertext. We then abstracted the problem of authentication for general messages, gave a solution to it, and proved that authenticated CPA-secure encryption provides a CCA-secure scheme.

But isn't it possible to combine encryption and authentication in other ways that achieve CCA-security? We will now see that some natural-looking combinations are in fact insecure – not only do they fail to achieve CCA-security, but they may even break the CPA-security of the original encryption scheme.

Encrypt-and-authenticate. The goal of encryption is to provide security, while the goal of authentication is to provide message integrity. This suggests that together, encryption and authentication should provide both security and integrity. Specifically, if (Enc, Dec) is CPA-secure and (Tag, Ver) is unforgeable against chosen message attack, will the following scheme be CPA-secure *and* unforgeable against chosen message attack?

$$Enc'(K, M) = (Enc(K_{\mathsf{CPA}}, M), Tag(K_{\mathsf{MAC}}, M))$$

In general, such a scheme is not only CPA-insecure, but could be insecure even against single message encryption! The fact that a MAC can be deterministic (in particular, the construction we gave was a deterministic ones), while CPA-secure encryption requires randomness already indicates that something is fishy. Suppose we want to CPA distinguish messages M and M'. We query the CPA oracle on M and compare the tag provided by the oracle with the tag provided to the distinguisher. If the tag is the same, the distinguisher can be confident that it is looking at an encryption of M and not of M'.

In fact, if the MAC we used was a bit different, this scheme would be even insecure for a single encryption. It could be that the tag of a message completely gives away the message: In fact, the scheme $Tag(K, M) = (M, F_K(M))$ is a perfectly valid MAC as long as F_K is a pseudorandom function, but it completely reveals the message! In this case there is clearly no security of any kind in the encryption.

Authenticate-then-encrypt. What if instead of authenticating the encryption, we first authenticate the message, and then encrypt the authenticated message?

$$Enc'(K, M) = Enc(K_{\mathsf{CPA}}, (M, Tag(K_{\mathsf{MAC}}, M)))$$
$$Dec'(K, C) = \begin{cases} M \text{ part of } Dec(K_{\mathsf{CPA}}, C), & \text{if } Ver(K_{MAC}, Dec(K_{\mathsf{CPA}}, C)) = 1 \\ \texttt{error}, & \text{otherwise.} \end{cases}$$

It is not difficult to see that if (Enc, Dec) is CPA-secure, then so is (Enc', Dec'). Intuitively, applying any function to a message (in particular, a MAC) should not affect indistinguishability of encryptions, since the ciphertext of a CPA-secure scheme gives negligible information about what was encrypted. More formally, if we can break the CPA-security of (Enc', Dec') then we can also break the CPA-security of (Enc, Dec) by having the new adversary/encryption oracle apply a tag before simulating the old adversary/encryption oracle. It is a good exercise to work out the details.

In general, however, (Enc', Dec') will not be CCA-secure.[1] Here is an example. Suppose (Enc, Dec) is a modified CPA-secure encryption scheme where Enc always applies a zero at the end of the ciphertext, and Dec ignores this zero in the decryption. It is easy to see that the resulting scheme is still CPA-secure. However, an adversary can now mount a chosen ciphertext attack: On a challenge ciphertext C, change the last bit of C from 0 to 1 and call the decryption oracle. This attack will reveal the original message (and its tag).

While this attack is contrived, one can imagine practical scenarios where the ciphertext contains some extra information that is ignored in the decryption – an end of file symbol, routing information, the header of an email. On the other hand the ciphertext attack sounds a bit unnatural: Why would a decryption algorithm agree to decode an incorrectly formatted ciphertext? There are more natural examples, where by merely obtaining the knowledge that a ciphertext was incorrectly formatted, the adversary can completely decrypt its challenge ciphertext.

Encryption and authentication with the same key. In our construction of CCA-secure encryption it was very important that we used *different* private keys for encryption and authentication. Using the same key for both can make the scheme insecure.

It is possible to give an example where our method for constructing CCA-secure encryption becomes insecure when the encryption key and the authentication keys are identical, but I don't know of a simple construction. Here is a direct construction that is based on similar ideas. This construction

[1] (Enc', Dec') *may* be CCA-secure for specific implementations of the CPA-secure scheme and the MAC used in the construction. However, it will not be so in general, which means that it is unsafe to use this design methodology for combining encryption and authentication.

is CCA-secure when instantiated with independent keys, but not even CPA-secure when the same key is reused twice:

$$Enc((K_1, K_2), M) = (S, F_{K_1}(S) + M, F_{K_2}(S + M)) \quad \text{where } S \text{ is a random string}$$

$$Dec((K_1, K_2), (S, C, T)) = \begin{cases} C + F_{K_1}(S), & \text{if } F_{K_2}(S + C + F_{K_1}(S)) = T \\ \texttt{error}, & \text{otherwise.} \end{cases}$$

In conclusion: Cryptographic components (in particular, encryption and authentication) cannot be combined in arbitrary ways. Such combinations may sometimes even be harmful and destroy the security properties of the original components.

3 Authenticating messages of arbitrary length

One annoying aspect of our definition of authentication is that it only works for messages of a given length m. What if the length of the message is not known in advance?

This is not merely a technical issue. To explain, let us go back to our original construction of MACs. To tag a message of length m, we used the scheme $Tag_m(M) = F_K(M)$, where $F_K\colon \{0,1\}^m \to \{0,1\}^k$ is a pseudorandom function.

Now suppose we want to tag a message M_1M_2 of length $2m$ (where M_1 are the first m bits and M_2 are the last m bits). To tag this message, we need a pseudorandom function F'_K that takes $2m$ bits of input. Suppose F_K was obtained by the GGM construction from some pseudorandom generator G. We can get F'_K by extending the construction F_K for another m levels, namely

$$F'_K(x_1 \dots x_{2m}) = G_{x_{2m}}(\dots G_{x_1}(K) \dots).$$

In particular, $F'_K(M_1M_2) = F_{F_K(M_1)}(M_2)$. Then the scheme $Tag_{2m}(M_1M_2) = F'_K(M_1M_2)$ (with the appropriate verification procedure) is certainly a secure MAC (against chosen message attack) for messages of length $2m$. But what happens if we use Tag_m and Tag_{2m} together? Consider the following attack: I first use the tagging oracle for Tag_m to obtain a tag $F_K(M)$ for some message $M \in \{0,1\}^m$. Now I can produce the tag $F'_K(MM) = F_{F_K(M)}(M)$, which is a forgery for Tag_{2m}! So it is possible to use short message tags in order to produce forgeries for long message tags.

One possible solution would be to use this scheme with an independently chosen key for every input length. However this is quite cumbersome. Without an a priori length on the messages that Alice and Bob want to authenticate, Alice and Bob will need infinitely long keys.

We will show an alternative solution which will allow authentication of arbitrarily long messages using private keys f fixed length. Before doing so, let us discuss the changes in the definition of MAC that need to be implemented to handle the variable-length case. In the functionality part, we now allow Tag and Ver to take arbitrary length messages as inputs – that is, they now have type $Tag\colon \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^t$ and $Ver\colon \{0,1\}^* \times \{0,1\}^t \to \{0,1\}$. The security part of the definition doesn't change. However, the change in the functionality does have an effect on security, because an adversary that tries to produce a forgery of length m is now allowed to invoke the tagging oracle on messages that are shorter or longer than m.

Since we already have a fixed-length message MAC, it makes sense to turn it into a variable-length message one. For simplicity, let's suppose we start with a MAC (Tag, Ver) for message length

k (that also produces tags of length k). Now say we want to authenticate a message of length $m > k$. A natural thing to try is to split this message into $\ell = m/k$ blocks $M_1 \dots M_\ell$ of length k and authenticate each block separately:[2]

$$Tag'(K, M) = (Tag(K, M_1), \dots, Tag(K, M_\ell))$$

Is this MAC secure against chosen message attack? A moment's thought shows that it isn't – in fact it is not even secure as a *fixed-length* scheme: Given the tag of $M_1 M_2$, we can produce the tag of $M_2 M_1$ by rearranging the blocks.

This attack suggests including some additional information that fixes the order of the blocks, like

$$Tag(K, M) = (Tag(K, (1, M_1)), \dots, Tag(K, (\ell, M_\ell)))$$

Now it is not possible to reorder blocks from the same message anymore. However, we can do an attack where we combine blocks from two different messages: If we see the tags of $M_1 M_2$ and $M_1' M_2'$, we can produce the tag of $M_1 M_2'$. To guard against this type of attack, we introduce a random identifier:

$$Tag'(K, M) = (S, Tag(K, (S, 1, M_1)), \dots, Tag(K, (S, \ell, M_\ell))) \quad \text{where } S \text{ is random.}$$

Under this scheme, it appears difficult to combine the tags of different messages. However there is still a type of attack that can be done: The tag of $M_1 M_2$ reveals the tag of M_1! To prevent this attack, we want to mark the last block of M with a special "end of message" symbol. One way to achieve this is to include an extra bit in each block which is set to 1 if this is the last block in the message, and 0 otherwise.

We have arrived at our candidate scheme for encryption of variable length messages. Let (Tag, Ver) be a MAC for message length $m = 3k + 1$. We construct a variable-length MAC (Tag', Ver') as follows:

$$\begin{aligned} Tag'(K, M) = (S, &Tag(K, (S, 1, M_1, 0)), Tag(K, (S, 2, M_2, 0)), \\ &\dots, Tag(K, (S, \ell, M_\ell, 1))) \quad \text{where } S \sim \{0,1\}^k \text{ is random.} \end{aligned}$$

$$Ver'(K, M_1 \dots M_\ell, (S, T_1, \dots, T_\ell)) = \begin{cases} 1, & \text{if } Ver(K, (S, i, M_i, 0)) = 1 \text{ for } 1 \le i \le \ell - 1 \\ & \text{and } Ver(K, (S, \ell, M_\ell, 1)) = 1 \\ \texttt{error}, & \text{otherwise.} \end{cases}$$

Here S, i, and M_i as represented by k bit strings.[3]

Claim 2. *If (Tag, Ver) is a $(O(s't), \varepsilon'/s' - s'/2^k)$ secure against chosen message attack, then (Tag', Ver') is (s', ε') secure against chosen message attack, where t is the size of the tagging circuit Tag.*

Proof. Let's assume (Tag', Ver') is not (s', ε') secure, so there exists an oracle circuit $A'^?$ of size s' such that $A'^{Tag'}$ produces a forgery with probablity ε'. Consider the following circuit $A^?$

[2]We'll assume m divides k; this can be handled at the message level by padding.

[3]Technically, this means Tag can only be used to sign messages up to length 2^k, but this is not a practical limitation.

A^T: Simulate the circuit $A'^?$. Whenever $A'^?$ calls its tagging oracle on input M, choose a random string S and answer its query by

$$(S, T(S,1,M_1,0), T(S,2,M_2,0), \ldots, T(S,\ell,M_\ell,1)).$$

when $A^?$ outputs a possible forgery $(M_1 \ldots M_\ell, (S, T_1, \ldots, T_\ell))$, choose a random i between 1 and ℓ and output $((S,i,M_i,0),T_i)$ if $i < \ell$ and $((S,i,M_i,1),T_i)$ if $i = \ell$.

Then A has size $O(s't)$. We now show that A^{Tag} produces a forgery with probability at least ε. Let S_j be the random string that $A^?$ chooses when the jth oracle call of $A'^?$ is made. We will show that

$$\Pr[A^{Tag} \text{ outputs a forgery}] \geq \frac{1}{s'} \Pr[A'^{Tag'} \text{ outputs a forgery and all } S_j \text{ are different}].$$

To see this, assume that $A'^{Tag'}$ outputs a forgery $F = (M_1 \ldots M_\ell, (S, T_1, \ldots, T_\ell))$ and all S_j are different. Since F is a forgery, $A'^?$ must not have queried its oracle on $M_1 \ldots M_\ell$. We claim that $A^?$ must not have queried its oracle on at least one of $(S,1,M_1,0), \ldots, (S,\ell-1,M_{\ell-1},0), (S,\ell,M_\ell,1)$. For if $A^?$ has made all these queries, then S must equal S_j for some j and because this S_j is unique, all these queries were made in the same round, and $A'^?$ must have queried $M_1 \ldots M_\ell$ in that round. It follows that at least one of $((S,1,M_1,0),T_1), \ldots, ((S,\ell-1,M_{\ell-1},0),T_{\ell-1}), ((S,\ell,M_\ell,0),T_\ell)$ is a forgery, so A^{Tag} must output a forgery with probability at least $1/\ell \geq 1/s'$ under these conditions.

To finish the proof, we calculate that

$$\begin{aligned}
&\Pr[A'^{Tag'} \text{ outputs a forgery and all } S_j \text{ are different}] \\
&\qquad\qquad\qquad \geq \Pr[A'^{Tag'} \text{ outputs a forgery}] - \Pr[\text{not all } S_j \text{ are different}].
\end{aligned}$$

By assumption, the first probability is at least ε'. We bound the second one as follows:

$$\Pr[\text{not all } S_j \text{ are different}] \leq \Pr[S_j \neq S_{j'} \text{ for some } j \neq j'] \leq \sum_{j \neq j'} \Pr[S_j \neq S_{j'}] \leq \binom{s'}{2} \cdot 2^{-k}$$

because there are at most $\binom{s'}{2}$ pairs (j, j'). Putting all the inequalities together we have that

$$\Pr[A^{Tag} \text{ outputs a forgery}] \geq \frac{1}{s'}\left(\varepsilon' - \binom{s'}{2} 2^{-k}\right) \geq \frac{\varepsilon'}{s'} + \frac{s'}{2^k}. \qquad \square$$