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. So it makes sense to apply a MAC to the ciphertext in order to prevent such forgeries.

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:

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

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

Here is the intuition for this construction: For the adversary to successfully take advantage of the decryption oracle, it 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 for the adversary to take advantage of the decryption oracle, it must query the oracle at some pair (C', T') where C' is different from C, which requires it to produce a forgery of an authenticated 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, so it can be removed. 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 easily 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 the adversary 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 the adversary (since it already knows the answer), so without loss of generality (at a small price in size) we shall be able to assume that the adversary never makes such a query.

Theorem 1. *If (Enc, Dec) is (s, ε) CPA-secure and (Tag, Ver) is $(s, \varepsilon/s)$ secure, 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} behaves exactly like $A_{\mathsf{CCA}}^{Enc_{\mathsf{CCA}}, Dec^*_{\mathsf{CCA}}}$, except that the decryption oracle always returns `error`. 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). (To simplify notation we omit the private keys.)

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_{MAC}^T: Choose a random key K_{CPA} and simulate $A_{\mathsf{CCA}}^{E',D'}(Enc(K_{\mathsf{CPA}}, M), T(Enc(K_{\mathsf{CPA}}, M))$, 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_{MAC}^T 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, so in particular the ith query made to D' was not given to T. So with probability at least ε/s, A_{MAC}^T produces a forgery (C_i, T_i) for (Tag, Ver). Since $A_{\mathsf{MAC}}^?$ has size $O(s'mt) \leq s$, (Tag, Ver) cannot be $(s, \varepsilon/s)$ 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 not give CCA-security, but they may even break the 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 encoding 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! In this case, the effect of encryption is completely destroyed by the MAC.

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. (While(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.) 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? However there are even 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. But here is a direct construction obtain following similar ideas. This construction is CCA-secure (I think) when instantiated with independent keys, but not even CPA-secure when the same key is reused twice:

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

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

You will explore the security of this construction in your homework.

In conclusion: Cryptographic components (in particular, encryption and authentication) cannot be combined in arbitrary ways, and such combinations may sometimes even be harmful and destroy the security 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: If Alice and Bob want to authenticate messages up to length m_{max}, it means that they must share a key of length km_{max}. It is easy to imagine m_{max} being very large, say when a laptop talks to its wireless router, so it would be better if we could avoid such long keys.

So let us try to develop an alternative solution. 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 for message length k (that also produces tags of length k). Such a MAC is obtained by applying a pseudorandom function $F_K\colon \{0,1\}^k \to \{0,1\}^k$ to the message. 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:[1]

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

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_1M_2, we can produce the tag of M_2M_1 by rearranging the blocks.

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

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

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_1M_2 and $M'_1M'_2$, we can produce the tag of $M_1M'_2$. To guard against this type of attack, we introduce a random identifier:

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

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

Under this scheme, it appears difficult to combine the tags of different messages. However there is still a type of attack that can be mounted: The tag of M_1M_2 reveals the tag of M_1! To prevent this attack, we want to mark the last block of M in a special way and require that it is tagged properly. One way to achieve this is to include an extra bit in each block which is set to 1 if we are looking at the last block, and 0 otherwise.

We have arrived at our candidate scheme for encryption of variable length messages. Let $F_K\colon \{0,1\}^{3k+1} \to \{0,1\}^k$ be a pseudorandom function family, and

$$Tag(K,M) = (S, F_K(S,1,M_1,0), F_K(S,2,M_2,0), \ldots, F_K(S,\ell,M_\ell,1)) \quad \text{where } S \text{ is random.}$$

The first three parameters of F_K are k bits long, and the last one is a single bit.[2] The verification $Ver(K,M,T)$ accepts iff T is a possible tag for M (for some S).

Claim 2. *If $\{F_K\}$ is an (s,ε) pseudorandom family, then (Tag, Ver) is $(\Omega(s), \varepsilon + O(t^2/2^k))$ secure against chosen message attack.*

Proof. Suppose (Tag, Ver) is not (s',ε') secure for $s' = \Omega(s), \varepsilon' = \varepsilon + (t^2+1)/2^k$, and let A be a circuit of size s' that produces a forgery with probability ε'. As usual, we first replace the pseudorandom function by a truly random one. Let $(RTag, RVer)$ be the same scheme but with a random function $R\colon \{0,1\}^{3k+1} \to \{0,1\}^k$. Then we know that

$$\Pr[A^{RTag} \text{ produces a forgery for } (RTag, RVer)] > \varepsilon' - \varepsilon$$

for otherwise we could obtain an $O(s')$-size circuit that ε-distinguished F_K from a random function by emulating A and outputting 1 if a forgery was produced. We will now show that this is impossible.

Suppose the ith query of A to its oracle is $M_{i1}M_{i2}\ldots M_{i\ell_i}$. The oracle $RTag$ returns

$$(S_i, R(S_i,1,M_{i1},0), R(S_i,2,M_{i2},0), \ldots, R(S,\ell,M_{i\ell_i},1)).$$

By a union bound, the probability that $S_i = S_j$ for some $i \neq j$ is at most $\binom{s'}{2}2^{-k} \leq s'^2 2^{-k}$. Intuitively, if all the S_is are different, the adversary should not be able to combine blocks from different calls to the oracle when producing its forgery. Suppose A^{RTag} produces a candidate forgery $(S, M_1, \ldots, M_\ell, T_1, T_2, \ldots, T_\ell)$. We consider two cases.

If $S \neq S_i$ for all i, if A^{RTag} outputs a forgery then at least T_1 should be of the form $R(S,\star)$. But since A has not seen any value of R of this form, all values of the form $R(S,\star)$ are independent of T_1, and so the probability that A^{RTag} outputs a forgery is at most 2^{-k}.

If, on the other hand, $S = S_i$ for some i and $S_i \neq S_j$ for all $i \neq j$, then for the forgery to be successful each T_i must be of the form $R(S,i,M_i,0)$, except for the last one which must be of the form $R(S,\ell,M_\ell,1)$ for some M_ℓ. So unless M equals M_i (the message queried in the ith round) there must be at least one block of the form $T_i = R(S,i,M_i,b)$ that was not returned by the oracle. In this case, T_i is statistically independent of $R(S,i,M_i,b)$, so the chance they are equal is at most 2^{-k}.

Putting everything together, by a union bound, the probability that A^{RTag} produces a forgery is at most $t^2/2^k$ (in case $S_i = S_j$ for some $i \neq j$) plus 2^{-k} (in the other case), and so

$$\Pr[A^{RTag} \text{ produces a forgery for } (RTag, RVer)] \leq (t^2+1)/2^k,$$

which contradicts the setting of parameters $\varepsilon' = \varepsilon + (t^2+1)/2^k$. □

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