Bob receives the following message:

```
help!  i am in danger.  wire 10000 dollars to account 4926017462 now.  -- alice
```

This is an atypical request from his good friend Alice. Yet she might be in trouble. What should Bob do?

An *authentication scheme* is a mechanism for Bob to verify that the message he purportedly received from Alice was indeed authored by Alice (authenticity), and it was not modified by Eve (integrity). Authenticity without integrity doesn't make much sense. Integrity without authenticity will be the topic of the next lecture.

To authenticate Alice as opposed to Eve, there must be some identifying feature that distinguishes the two. In a telephone call this would be Alice's distinctive voice. In a letter it would be Alice's unique handwriting. In the digital world, Alice's identifying feature is her secret key.

This lecture is about private-key authentication. Alice's secret key is known to Bob only. Eve has no information about it. Alice's goal is to send a message of her choice to Bob. Bob can verify that Alice's message is authentic. Eve, on the other hand, should not be able to *forge* an authentic message. Messages crafted by Eve do not pass Bob's verification.

Secrecy (as guaranteed by secure encryption) and authenticity are incomparable goals. Authentication has no secrecy requirement. The content of Alice's message need not be hidden from Eve. Conversely, none of the encryption schemes we have seen so far guarantee authenticity. We will explain why shortly.

1 Message authentication codes

In a Message Authentication Code (MAC), Alice appends an authentic *tag* to her message. Bob runs a verification algorithm that certifies the message-tag pair is valid (see Figure 1).

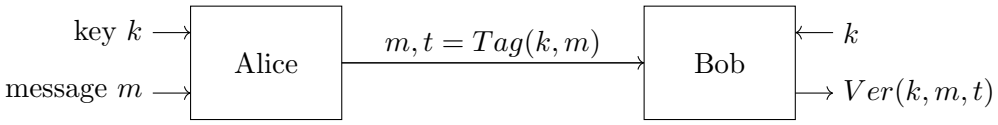


FIGURE 1: Message authentication.

A MAC thus consists of two algorithms: A *tagging algorithm* that, given a secret key k and a message m, outputs a tag t for m; and a *verification algorithm*, that given the same key k and a message-tag pair (m, t), accepts or rejects. The functionality requirement of a MAC is that verification always accepts correctly tagged messages:

$$Ver(k, m, Tag(k, m)) \text{ always accepts.}$$

In the honest operation of a MAC, Alice would be tagging any number of messages of her choice. Bob would then accept all message-tag pairs as authentic as in Figure 2.

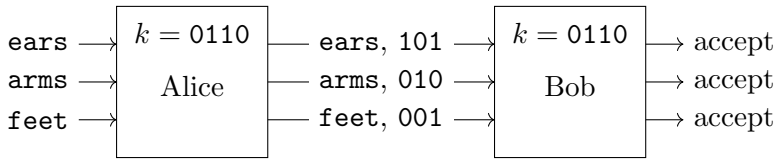


FIGURE 2: Honest operation of a MAC.

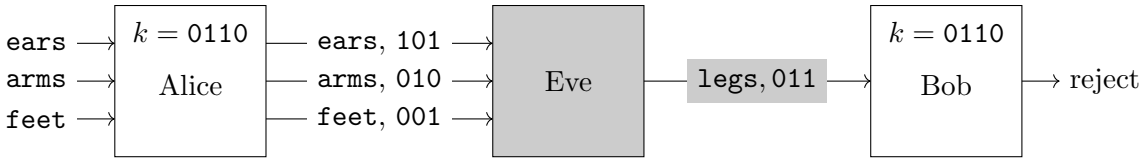


FIGURE 3: Unforgeability of a MAC.

To understand security, imagine Eve as a woman in the middle. She intercepts all message-tag pairs from Alice. Her goal is to create a *forgery*: A tagged message that was not produced by Alice, but verifies as authentic by Bob. The MAC is *unforgeable* if Eve's chance in producing such a pair is close to nil. See Figure 3. With this setup in mind it is instructive to see three non-examples.

Checksum. Checksum is a common mechanism for error detection. For example, if each letter was represented by its ASCII encoding, the checksum of `love` would be

$$\texttt{l} + \texttt{o} + \texttt{v} + \texttt{e} = \texttt{01101100} + \texttt{01101111} + \texttt{01110110} + \texttt{01100101} = \texttt{00010000}.$$

The checksum is a lousy MAC for the same reason that Caesar's cipher is a lousy encryption. It is deterministic. Eve can calculate checksums just as easily as Alice can. If all correct checksums pass Bob's verification, Eve can "forge" the checksum of any message of her choice.

Randomized checksum. What if the checksum is masked with the secret key, i.e., $t = \text{checksum}(m) + k$? Eve will now have difficulty creating a consistent checksum without knowing the key. She can, however, still forge a valid tag. After intercepting any message-tag pair (m, t), Eve can recover the key as $k = t + \text{checksum}(m)$. Once Eve gets hold of the secret key, she can forge a tag for any message of her choice.

There is nothing special about checksum: If Alice replaces it with any "feature" $\phi(m)$ of the message—even a complete copy of the message—Eve can forge in the same manner: First, she extracts the key as $k = \phi(m) + t$, and then she forges a signature of any message $m^\star$ of her choice as $\phi(m^\star) + k$.

Encryption. To prevent Eve from learning the key, how about encrypting the feature $\phi(m)$ with a CPA-secure scheme like the PRF pad? The tag would now look like $(x, \phi(m) + F(x))$. Here F is a PRF whose key is shared among Alice and Bob, and x is a random pointer. Eve can no longer extract the key. She can still forge. Suppose she intercepted message $m = \texttt{love}$ from Alice with tag $(x, t = \phi(\texttt{love}) + F(x))$. The tag $(x, t + \phi(\texttt{love}) + \phi(\texttt{hate}))$ is then a valid tag for $m^\star = \texttt{hate}$. Eve can produce it on her own without knowing the secret key to F. Eve has turned a valid tag for `love` into a valid tag for `hate`. This is forgery.

There is nothing special about the PRF pad here. None of the encryptions we have seen so far are suitable for authenticating messages.

The Pseudorandom Function MAC

In the last two examples, Eve was able to *learn* what a valid tag of `hate` looks like by observing a valid tag of `love`. For the MAC to be unforgeable, Eve should not be able to tag any new message from observing previous message-tag examples. One way to accomplish this is to make tags of different messages look like independent random strings. That is precisely what a pseudorandom function does.

The Pseudorandom Function MAC (PRF MAC):

Shared key: The key k used to set up PRF F.

Tagging: Alice attaches the tag $t = F(m)$ to message m.

Verification: Upon receiving the message-tag pair (m, t), Bob accepts if t equals $F(m)$, and rejects if not.

The PRF MAC is clearly functional. To understand its (un)forgeability, let us first ignore Alice altogether. If Eve hasn't intercepted any messages from Alice, she needs to produce a forgery out of thin air. Suppose she wants to forge a tag to $m^\star = \texttt{legs}$. The only way she can fool Bob into accepting is by figuring out the value $t^\star = F(\texttt{legs})$. Eve does not know the key to F. If F is a (pseudo)random function, she can do no better than guess what $t^\star$ is. If tags are ℓ bits long, her chance of getting it right is $2^{-\ell}$.

This attack reveals yet another difference from encryption: Authentication can never be perfectly secure. There is always a valid tag for every message, and Eve can guess it with probability $2^{-\ell}$ on any one attempt. This probability is tiny when, say, $\ell = 128$. As usual, the objective is then not to make forgeries impossible, but merely extremely improbable.

Let's now bring Alice back into the game. Suppose Eve intercepted some message-tag pairs output by Alice, like $\texttt{ears}, F(\texttt{ears})$ and $\texttt{arms}, F(\texttt{arms})$. Do those help her forge $t^\star = F(\texttt{legs})$? If F were a truly random function, $F(\texttt{ears})$ and $F(\texttt{arms})$ are nothing but sequences of independent random bits. They do not at all affect the value $F(\texttt{legs})$, or anything else for that matter. Eve did not learn anything useful from the intercepted tags. Her effective probability of guessing $F(\texttt{legs})$ correctly remains about $2^{-\ell}$.

2 MACs for long messages

One drawback of using a PRF as a MAC is that it can only authenticate fixed-length messages. The message must fit the PRF's input. A typical PRF or cipher might take in 128 bits of input. Typical messages like documents or videos are orders of magnitudes bigger.

A sensible solution is to split the message into fixed-size blocks and apply the PRF MAC to each block separately. There are secure and insecure ways to execute this transformation. For concreteness, assume the message m splits into a left block m_L and a right block m_R.

Tag the checksum $m_L + m_R$. Tag t verifies as authentic when $t = F(m_L + m_R)$. This is insecure. Eve can forge m' from any m that has the same checksum, without touching the tag. For example, $(\texttt{levelogs}, t)$ is a forgery of $(\texttt{lovelegs}, t)$ whenever t is a valid tag, i.e. $t = F(\texttt{love} + \texttt{legs})$.

Tag m_L and m_R separately. The tag of $m_L m_R$ is the concatenation $F(m_L)F(m_R)$. (This is the analogue of ECB mode for encryption.) Tag $t_L t_R$ verifies as authentic when $t_L = F(m_L)$ and $t_R = F(m_R)$. Eve can now forge by swapping the messages and tags. If $(\texttt{lovelegs}, t_L t_R)$ is a valid pair, so is $(\texttt{legslove}, t_R t_L)$. She can also "replay" a message chunk twice. $(\texttt{lovelove}, t_L t_L)$ is another forgery.

Register the order. The tag of $m_L m_R$ is the concatenation $F(m_L\texttt{0})F(m_R\texttt{1})$. Bob accepts only when $t_L = F(m_L\texttt{0})$ and $t_R = F(m_R\texttt{1})$. Swapping no longer produces a valid forgery. Eve can, however, still forge. After observing the MACs $t_L t_R$ and $t'_L t'_R$ for two messages $m = \texttt{lovelegs}$ and $m' = \texttt{hatearms}$, she can produce the forgery $\texttt{lovearms}$ with valid tag $t_L t'_R$. Indeed, t_L equals $F(\texttt{love0})$ and t'_R equals $F(\texttt{arms1})$.

The last attack illustrates a difficulty in concatenating deterministic stateless MACs. Forgeries can be obtained by mixing up message-tag pairs from different sessions. One solution is to encode the identity of the session by a unique session number. This requires keeping state. A stateless alternative is to pick the session identifier at random. This is the MAC analogue of encryption in counter mode.

Counter-Style PRF MAC:

Shared key: The key k used to set up PRF F.

Tagging: Given an n-block message $m_1 m_2 \dots m_n$, Alice picks a random session ID x. She tags m by $(x, F(x, 1, m_1), F(x, 2, m_2), \dots, F(x, n, m_n))$.

Verification: Upon receiving $(m_1 \dots m_n, x, t_1 \dots t_n)$, Bob accepts if $F(x, 1, m_1) = t_1$, $F(x, 2, m_2) = t_2$, and so on, and rejects otherwise.

As long as the session IDs of all the intercepted messages are distinct, Eve has no reasonable forging strategy. Since F is pseudorandom, she is effectively constrained to rearranging message-tag pairs intercepted from Alice. The session and block numbers ensure that the only valid ordering is the one produced by Alice.

Should some two session IDs match by chance, however, Eve can mix and match the message-tag pairs into a forgery. How likely is this to happen? If the session ID is ℓ bits long, the probability that a given pair of sessions has matching IDs is $2^{-\ell}$. If Eve intercepts q sessions, there are $\binom{q}{2} = q(q-1)/2$ potential pairs with matching IDs. By the union bound, the probability that at least one of these IDs match is at most the sum of the probabilities for each of the $\binom{q}{2}$ candidate session pairs, namely $\binom{q}{2} \cdot 2^{-\ell}$. If, say, $\ell = 128$ and q is in the millions, session IDs will be unique except with probability less than 2^{-80}.

CBC-MAC

One major drawback of the counter-mode PRF MAC is its long tag. Cipher Block Chaining MAC, or CBC-MAC, is the MAC analogue of CBC Encryption Mode. Message $m = m_1 m_2 m_3 \cdots m_n$ is tagged by

$$CBCMAC(m) = F(\cdots F(F(F(m_1) + m_2) + m_3) \cdots + m_n) \tag{1}$$

Here, F is a PRF with equal input and output length.[1] See Figure 4. To verify (m, t), Bob checks that $CBCMAC(m)$ matches t.

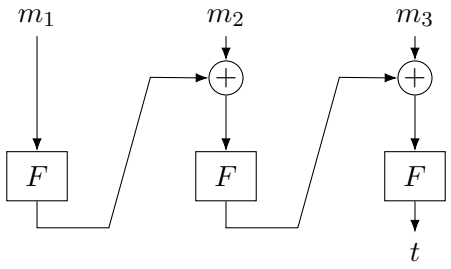


FIGURE 4: CBC-MAC.

The length of the tag here is independent of the length of the message being tagged. It is equal to the output length ℓ of the underlying PRF, typically a few hundred bits.

To get a sense of security, suppose Eve observes the tag t of $m_L m_R =$ `lovelegs` and tries to reorder them. Her attempted forgery is `lovelegs`$, t$. Bob would not accept unless $CBCMAC($`lovelegs` and $CBCMAC($`legslove` match, or $F(F($`love`$)+$`legs`$) = F(F($`legs`$)+$`love`$)$. The output to the outer F are unlikely to match unless their inputs do, namely $F($`love`$)+$`legs`$ = F($`legs`$)+$`love`. Equivalently, $F($`love`$)$ must equal $F($`legs`$)$ plus the fixed value `love`$+$`legs`. This is extremely unlikely for a (pseudo)random F.

MACs for variable-length messages

As described, Counter-mode PRF MAC and CBC-MAC are secure as long as Bob knows the length n of the message being tagged (in blocks). If Bob does not know n in advance, Eve can excise a suffix of the message yet pass verification. In counter style, for example, a two-block message $m_1 m_2$ is tagged by $(x, F(x, 1, m_1), F(x, 2, m_2))$. Eve can remove all parts pertaining to m_2 and still end up with the valid message-tag pair $m_1, (x, F(x, 1, m_1))$.

Alice can thwart this type of attack by prepending the message m by its length n. That is, the MAC is applied to the string n followed by $m_1 \ldots m_n$. The length n is then authenticated as part of the message. If Bob detects an inconsistency between the declared and actual length upon verification, he rejects. There are alternative mechanisms for messages whose length is not known in advance.

[1] In the analogous encryption mode, F was a PRP. Bob had to invert it to decrypt.

3 The perils of decryption errors

In Lecture 4 we pointed out a vulnerability of the PRF pad when Eve can interfere with the communication between Alice and Bob. This Eve-in-the-middle could intercept an encryption of `love` and turn it into an encryption of `hate` without knowing anything about the secret PRF key. None of the encryptions we have seen so far can withstand a woman in the middle attack.

There are realistic scenarios in which a woman in the middle can completely break encryption. Here is a contrived but illustrative example. Suppose Alice is in the middle of an online purchase and needs to send Bob her credit card number. Bob received the ciphertext, decrypts it, and recovers the message `7294 0283 5831 love`. This is not a valid credit card number. What should he do?

In software engineering we learn that if a program receives an unexpected input, it should return an error. An error message produced by a decryption algorithm, however, is far from innocuous. Eve can exploit it to gain information about the underlying message.

To be specific, suppose Eve intercepts a PRF pad encryption (x, c) of the first digit m in Alice's credit card number. Eve mauls the ciphertext into $(x, c+9)$ and passes it on to Bob. As c equals $F(x)+m$, Bob decrypts to $m+9$. Unless m equals zero, this is not a valid digit. Bob then announces an error. If m were to equal zero, decryption would have proceeded as expected. The presence of an error allows Eve to tell whether m equals zero or not.

Should an error have occurred, Eve can try again with $(x, c+8)$. She can now tell if m equals one or not. Challenging Bob digit by digit, Eve can eventually recover all of Alice's credit card number. All she did was observe which decryptions are faulty and which are not.

Padding and decryption errors Actual encryption standards have been broken using similar attacks. A famous case in point is PKCS #7 for CBC mode encryption. One practical issue in modes of encryption is that the message might not break up exactly into blocks. Part of the last block might be unused. For example, if the blocks are 4 bits long and the message is `0110 10`, what should Alice write in the remaining slots? One possibility is to fill them with zeros. But then Bob cannot disambiguate between the messages `0110 1`, `0110 10`, `0110 100`, `0110 1000`.

PKCS #7 is a padding standard that fills up the unused message slots in an unambiguous way. Messages are represented by a sequence of *bytes*, and blocks are 16 bytes long. If, say, the last 5 bytes of the last block are unused, PKCS #7 fills them in with the pattern `55555`. Bob now knows exactly where the message ends and the padding begins.

What happens if the padding is malformed, say it ends in `65555`? PKCS #7 prescribes that Bob should declare an error. With a clever mauling strategy, Eve can completely recover Alice's message from the pattern of decryption errors only!

4 Authenticated Encryption

The objective of authenticated encryption is to prevent Eve from either

(a) learning anything about Alice's messages, or

(b) goading Bob into decrypting a message that was not encrypted by Alice

by controlling the communication between Alice and Bob. It can be accomplished by combining encryption and MACs judiciously. There are three strategies to try. In all cases, encryption and authentication are applied with independent keys. The effective shared key length is the sum of the two.

Encrypt-and-authenticate

Alice sends an encryption of her message, followed by a MAC:

$$(Enc(m), Tag(m)).$$

Upon receiving the ciphertext-tag pair (c, t), Bob decrypts c to m. He declares an error if t is an invalid tag for m. See Figure 5.

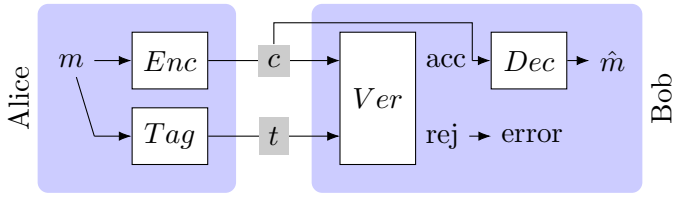


FIGURE 5: Encrypt-and-authenticate

This combination is not secure in general. With a deterministic MAC, it can reveal information that the encryption alone would hide. To understand why, suppose encryption is done with the PRF pad and authentication is done with the PRF MAC. Alice encrypts m into $x, F_e(x) + m$ and tags it with $F_a(m)$. Eve observes the ciphertext-tag pair $(x, F_e(x) + m, F_a(m))$.

Suppose that now Eve observes a pair of encryptions

$$(x, F_e(x) + m, F_a(m)) \qquad \text{and} \qquad (x', F_e(x') + m', F_a(m'))$$

She can then deduce whether m and m' are equal or not. If they are equal, the tags $F_a(m)$ and $F_a(m')$ are certain to match. If they are different they are unlikely to match. Encrypt-and-authenticate ruins the CPA security of encryption alone.

Encrypt-then-authenticate

Alice encrypts the message and then tags the ciphertext:

$$(Enc(m), Tag(Enc(m))).$$

Upon receiving ciphertext-tag pair (c, t), Bob declares an error if t is an invalid tag for c. Otherwise he decrypts c. See Figure 6.

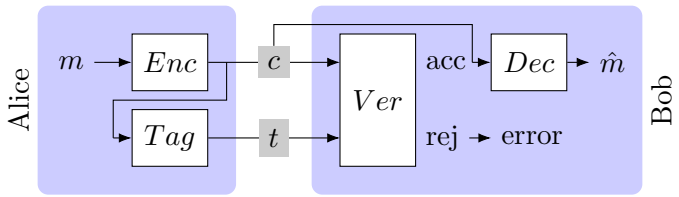


FIGURE 6: Encrypt-then-authenticate.

With independent keys, CPA-secure encryption, and a strongly unforgeable MAC applied to the complete ciphertext (including its nonce or IV), this is a secure way to implement authenticated encryption. Here is why. Suppose Eve-in-the-middle intercepts some ciphertexts c and their MACs t. As the MAC is unforgeable, Eve is unable to produce a new ciphertext–tag pair that passes verification. Any attempt to do so will result in erroneous decryption. The only messages Bob will decrypt without error are those that were prepared by Alice.

Can Eve learn anything about Alice's messages? As long as Enc is CPA-secure, the intercepted ciphertexts hide the corresponding plaintexts from her. The tags reveal no additional useful information; Alice derived them from the ciphertexts without looking at the message. The only thing Eve can do with these random-looking ciphertext-tag pairs is to submit them to Bob and wait for a possible decryption error. But she already knows that a decryption error will not be triggered as all the tags for these ciphertexts are valid. In short, Eve learned nothing new by looking at Alice's intercepted data, and she learned nothing new from triggering decryption errors on Bob's side.

To get a concrete illustration, Alice can use the PRF pad for encryption and the PRF MAC for authentication. The combination is a secure authenticated encryption for short messages.

PRF-based authenticated encryption:

Shared keys: Keys e and a to PRF instantiations F_e and F_a.

Encryption: To encrypt m, Alice picks a random pointer x and sends $(x, F_e(x) + m, F_a(x, F_e(x) + m))$.

Decryption: Upon receiving x, c, t, Bob declares an error if $t \neq F_a(x, c)$. Otherwise, he recovers the message $F_e(x) + c$.

Authenticate-then-encrypt

Alice tags the message and then encrypts the message-tag pair:

$$Enc(m, Tag(m)).$$

Bob decrypts the message-tag pair (m, t), and declares an error if t is an invalid tag for m. See Figure 7.

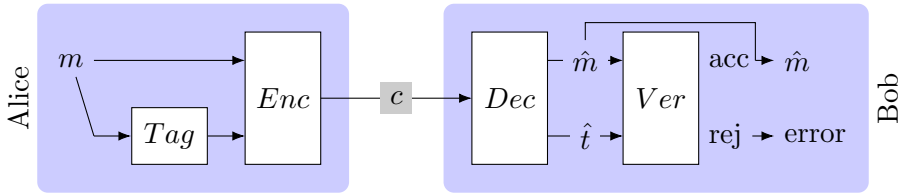


FIGURE 7: Authenticate-then-encrypt.

Authenticate-then-encrypt is not always a secure combination, but it often happens to be secure in practice. Its security depends on the implementation details of the underlying encryption and authentication schemes. It has to be analyzed case by case.

If we instantiate encryption with the PRF Pad and authentication with the PRF MAC, ciphertexts have the form

$$x, F_e(x + (m, F_a(m)))$$

This instantiation is a secure authenticated encryption. In the encryptions that Eve observes, the pointers x are likely to be distinct, and the PRF F_e is likely to be evaluated on distinct inputs. From Eve's perspective they look like independent random bits. If Eve tries to forge by mixing their respective parts, the mismatched input to F_e is unlikely to authenticate. Bob is likely to detect the attempted forgery and Eve has not learned anything. If she tries to create a new encryption out of thin air, it is unlikely to pass verification for the same reason.

A related construction that is secure for long messages is Counter with CBC-MAC (CCM). The tag t is derived using (1). The message-tag pair is then encrypted together in counter mode. The ciphertext has the form

$$x, F(x + 1) + m_1, \ldots, F(x + n) + m_n, F(x + n + 1) + CBCMAC(m).$$

If Eve intercepts Alice's messages, she receives no information from them. The construction can be analyzed to show that unintended collisions among the inputs to F, including the ones inside CBCMAC, are unlikely to occur.[2] If this is the case, Eve cannot trigger any informative decryption errors.

[2] Although in general encryption and authentication should use different keys, it turns out that there is no harm in using the same F with the same key for encryption and authentication in this particular scheme.

5 Summary

Message authentication is a mechanism that underlies both authenticity and integrity of transmissions. Authenticity means that Bob is sure the message originated from Alice. Integrity means that Alice's message was not tampered with. Secrecy is not a goal of message authentication.

MACs are a common mechanism for authenticating messages under a shared key. The value of a PRF with long output is one example. Bob can verify it, but Eve cannot guess it. For long messages, Eve can try to forge by rearranging message-tag blocks or by repeating blocks. Counter-style PRF MAC and CBC-MAC are designed to thwart such attacks.

When the unauthenticated data is a ciphertext, Eve-in-the-middle can potentially maul it into the encryption of a different message. She can also try to gain information by modifying it in various ways and observing which modifications give rise to decryption errors. Authenticated encryption is designed to prevent both attacks. It can be achieved generically by encrypting using any CPA-secure scheme then authenticating the ciphertext. Other combinations have questionable security, or are outright insecure.