Digital signatures are a public-key mechanism for message authentication. A (digital) signature allows anyone who knows Alice's public key to verify that a message was produced by Alice. The steps of a digital signature are

1. **Key generation:** Alice runs the randomized key generation algorithm $Keys$ to produce a pair of keys: A secret (signing) key sk and a public (verification) key pk. She keeps sk to herself. She posts pk for anyone to see.
2. **Signing:** To sign a message m, Alice runs $Sign_{sk}(m)$ and attaches its output as a signature.
3. **Verification:** Upon receiving message-signature pair m, s, from Alice, Bob runs $Ver_{pk}(m, s)$ with Alice's public key pk to verify that the pair is valid.

The functionality requirement is that actual signatures pass verification: For every message m and every key pair (sk, pk), $Ver_{pk}(m, Sign_{sk}(m))$ accepts.

The security requirement is very similar to the one for message authentication: An Eve-in-the-middle who intercepts arbitrary message-signature pairs cannot create a forgery, namely a fresh message m^* with a "signature" s^* that verify as correct. The probability that Eve can make $Ver_{pk}(m^*, s^*)$ accept is extremely small. The only difference from MACs is that Eve knows Alice's public key pk. She can therefore test whether her forgery is successful or not.

The public verifiability of signatures has an important consequence. Signatures are *transferable*. If Alice signs a message intended for Bob, then Bob can prove to Charlie that the message was produced by Alice. Charlie runs the same verification algorithm with Alice's public key. In contrast, if the message was authenticated with a MAC under a shared secret key, only the holder of the secret key, namely Bob, can be convinced of its authenticity.

1 Signatures versus public-key encryption

Signature schemes, like public-key encryption, are asymmetric. In public-key encryption Alice encrypts with Bob's public key and Bob decrypts with his secret key (Figure 1(a)). In signature schemes key ownership is reversed among sender and receiver. It is Alice that signs with her secret key, while Bob verifies with her public key (Figure 1(b)). When the notion of a digital signature was first proposed—by none other than Diffie and Hellman—they suggested building one by "reversing" public-key encryption. We now know that this analogy is misleading. It is instructive to understand why.

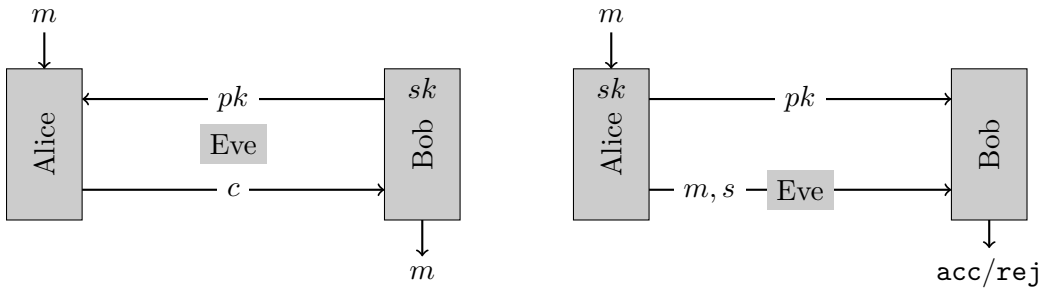

FIGURE 1: Public-key (a) encryption versus (b) signatures.

Example 1: Suppose the key pair is sampled as for El Gamal encryption. The secret key is an exponent X and the public key is $pk = g^X$ modulo a safe prime p. (As usual g is a quadratic residue and X is taken

modulo $(p-1)/2$. That is not particularly relevant here.) Now Alice signs her message with the secret key and Bob verifies that the signature is consistent with her public key:

$$Sign_X(m) = X \qquad Ver_{pk}(m, X) \text{ accepts if } g^X = pk.$$

This scheme is insecure for two reasons. First, Alice sends out her secret key in the clear. Upon intercepting it Eve can forge Alice's signature of any message she chooses. Second, nothing here binds the message to the signature. Eve can replace Alice's message with any other message she chooses. The signature will still verify as correct.

Example 2: Now the key pair e, d is sampled as for RSA. Alice signs with the decryption key and Bob verifies with the encryption key:

$$Sign_d(m) = m^d \bmod n \qquad Ver_{pk}(m, s) \text{ accepts if } m = s^e \bmod n.$$

Signatures no longer obviously reveal the secret key, and the signature now depends on the message. Distinct messages in fact always produce distinct signatures (assuming m is coprime to n).

Nevertheless, the signature scheme is insecure. For example, $(1, 1)$ is always a valid forgery. So is any message-signature pair of the form $(s^e \bmod n, s)$. Eve does not need to intercept anything to create one.

There is another type of forgery Eve can produce. Suppose she intercepts message-signature pair (m, s) from Alice. Then the pair (m^2, s^2) is a forgery (unless $m = 1$). More generally, given any pair of signatures (m_1, s_1), (m_2, s_2), Eve can produce the forgery $(m_1 m_2, s_1 s_2)$. The forgery will be accepted because $(s_1 s_2)^e = s_1^e s_2^e = m_1 m_2$.

Even though these proposals are problematic, they exemplify important lessons about signatures.

2 Chained one-time signatures

The scheme in Example 1 does satisfy a rudimentary form of security: If Eve intercepts no messages from Alice she is unable to produce a forgery. That would require her to calculate the discrete log of Alice's public key. This scheme can be upgraded into a fully fledged signature provided we make an unrealistic assumption. The assumption is that there are only two possible messages Alice may send to Bob: the bits `0` and `1`.

One-bit signature (CDH-based):

Key generation: Alice samples a pair of private exponents X_0, X_1 as her secret key. She posts the pair $pk_0 = g^{X_0}, pk_1 = g^{X_1}$ as her public key.

Signing: Alice signs bit m as X_m.

Verification: Upon receiving (m, X), Bob accepts if pk_m equals g^X, and rejects if not.

The one-bit signature is clearly functional. It also happens to be secure. If Eve hasn't intercepted anything from Alice, the only way she can forge a signature of anything is by extracting one of the secret keys from the respective public key. She either needs to extract X_0 from $pk_0 = g^{X_0}$ or X_1 from $pk_1 = g^{X_1}$. Namely, she has to take one of two discrete logs. This is intractable for her.

Interceptions don't help Eve at all. If, for instance, she intercepts the message-signature pair $(0, X_0)$, it no longer counts as a forgery. To obtain one she is still left with calculating the discrete log of pk_1. If she intercepts both $(0, X_0)$ and $(1, X_1)$ she cannot forge because there are no messages left for her to sign!

Interlude: One-way functions

The security of the one-bit signature relies on a generic property of the modular exponentiation function $f(X) = g^X \bmod p$. This function is easy to evaluate, but difficult to invert.

Such functions f are called *one-way*. Given x, $f(x)$ can be calculated efficiently. Given an output y obtained by evaluating f on a random input x, recovering x, or any other input x' that hits y, is intractable for Eve.

The one-bit signature can be based on any one-way function. Modular exponentiation is merely one candidate one-way function. Other plausible candidates are SHA-3, Trivium (as a function of the secret key) and AES (as a function of the secret key evaluated on any input).

In fact, any pseudorandom generator, viewed as a function of its seed, is a one-way function. If there is a way to recover the seed from the output, then its output can be distinguished from random. A truly random output is unlikely to be consistent with any possible seed, so recovery would fail on it.

All collision-resistant hash functions are also one-way. This happens for an interesting reason. To be specific, suppose h maps 512 input bits into 256 output bits. One collision-finding strategy for Eve is to start with a random x, calculate $y = h(x)$, and try to invert h on y. Won't she then end up with x which is exactly where she started?

No. If Eve succeeds in inverting, she is extremely likely to end up with a different preimage x' of y. The pair (x, x') is then a bona fide collision. The reason is that h shrinks 2^{512} possible inputs onto 2^{256} possible outputs. On average, an output y will arise from $2^{512}/2^{256} = 2^{256}$ possible inputs. The inverter does not see x itself but only $y = h(x)$. The actual x is equally likely to have been any one of these 2^{256} possibilities. If the inversion succeeds, it is extremely unlikely to give back the actual x that Eve picked.

The abundance of one-way functions puts Lamport's one-bit signature on solid foundations. There are many potential one-way functions that can be used to implement it. If modular exponentiation is one day found to be efficiently invertible, SHA-3 can take its place. If, by some miracle, SHA-3 is broken, there is always AES.

One-time signatures

Signing a one-bit message is not particularly impressive. A longer message can be signed bit by bit, using separate keys for each bit.

Lamport's one-time signature for n-bit messages:

Key generation: Alice samples n pairs of private "inputs" $X_0(1), X_1(1), X_0(2), X_1(2), \ldots, X_0(n), X_1(n)$ as her secret key. She posts the $2n$ values $pk_b(i) = f(X_b(i))$ as her public key.

Signing: Alice signs message $m = m_1 \ldots m_n$ bit-by-bit as $X = X_{m_1}(1) \cdots X_{m_n}(n)$.

Verification: Upon receiving (m, X), Bob accepts if $pk_{m_1}(1) = f(X_{m_1}(1))$ and $pk_{m_2}(2) = f(X_{m_2}(2))$ and so on up to $pk_{m_n}(n) = f(X_{m_n}(n))$.

For instance, the signature of message `010` is the bit sequence $X_0(1)X_1(2)X_0(3)$. Bob verifies that $f(X_0(1)) = pk_0(1)$ and $f(X_1(2)) = pk_1(2)$ and $f(X_0(3)) = pk_0(3)$.

Lamport's one-time signature is functional but insecure. Upon intercepting the message-signature pairs $(\texttt{00}, X_0(1)X_0(2))$ and $(\texttt{11}, X_1(1)X_1(2))$, Eve can mix them to produce the forgery $(\texttt{01}, X_0(1)X_1(2))$.

To produce such a forgery, Eve must intercept at least two of Alice's messages. Had she only intercepted, say, $(\texttt{00}, X_0(1)X_0(2))$, she would be unable to forge. The potential forged message must contain bit `1` in either position. The corresponding forged signature X^* must map to $pk_1(1)$ or $pk_1(2)$ under f, depending on the position. But Eve cannot find this X^* without inverting f.

In short, Lamport's signature is secure provided that Eve intercepts at most one of Alice's signed messages.

Chaining

One-time unforgeability is a very weak security requirement. Nevertheless, a one-time unforgeable scheme can be upgraded into a fully secure one provided that Alice and Bob maintain state.

The idea of chaining is that every time Alice signs a message, she attaches to it a fresh public key in preparation for her next message. At any given time, Alice's state consists of a secret key sk for a one-time signature. Bob's state consists of the corresponding public key pk. These are initialized as in Lamport's one-time signature scheme.

Now suppose Alice wants to sign message m. As part of her signing algorithm, she first runs key generation to sample a fresh key pair sk', pk'. She then sends the message m, pk' to Bob, signed with her current secret key sk:

$$s = Sign_{sk}(m, pk').$$

After she sends her message, Alice forgets sk and updates her secret key to sk'. Upon receiving the message (m, pk') and signature s, Bob validates the pair under his current public key pk.

Chaining one-time signatures preserves their security. Bob verifies every message under a different (independent) public key. As long as each such public key is authentic, Eve cannot forge signatures verified by it because she hasn't intercepted any relevant message-signature examples. Her only remaining strategy is to forge the public key itself. The corresponding secret key was only used once. She cannot succeed unless she breaks the underlying one-time signature.

This method of chaining signatures has a significant drawback. Suppose messages are 256-bits long. The public key in Lamport's one-time (CDH-based) signature then consists of $2 \cdot 256 = 512$ quadratic residues. When Alice sends her first message, she needs to sign not only the message itself, but also a fresh public key. The effective message length has grown from 256 bits to $512 \cdot b$ bits, where b is the output length of the one-way function f (in this example, the bit length of a quadratic residue). The associated public key is now $(512 \cdot b)^2 = 2^{20 \cdot 2}$ bits long. After, say, 10 iterations, the keys become $(512 \cdot b)^{10}$ bits long. This is unreasonable.

Alice and Bob can keep the key length under control with a collision-resistant hash. The signature is applied to a fixed-length hash h, say 256 bits, of the message–public-key pair m, pk'. The key length drops to $256b$ bits at every iteration.

Stateful chained one-time signature:

Key generation: Alice samples a key pair (sk, pk) for a one-time signature $(Sign, Ver)$ for 256-bit messages. She sends pk to Bob.

Signing: Alice samples a fresh key pair (sk', pk'). She sends $m, pk', Sign_{sk}(h(m, pk'))$ to Bob. She refreshes her secret key to sk'.

Verification: Upon receiving (m, pk', s), Bob accepts if $Ver_{pk}(h(m, pk'), s)$ accepts. If verification succeeds he refreshes Alice's public key to pk'.

Assuming h is collision-resistant the chained signature remains unforgeable. With additional ideas, this chained one-time signature can be made stateless.

SPHINCS+ is a practical signature scheme based on chaining one-time (or few-time) signatures. Compared to variants we will see shortly, it is relatively inefficient. Its main advantage is that its security is based only on the collision-resistance of its underlying hash function. The same hash function is used to instantiate f in the underlying one-time signature.

3 The full domain hash

Eve is able to forge plain RSA signatures as in Example 2 owing to the multiplicative structure of RSA. This structure is destroyed by applying a "random-looking" hash h before exponentiation. The hash h

takes an arbitrary message m as its input and produces an output within the RSA modulus n.

Hash-then-RSA:

Key generation is as in RSA: Alice samples primes p, q, chooses $ed \equiv 1 \bmod (p-1)(q-1)$, publishes $n = pq$ and e, and keeps d as her secret key.

Signing: Alice signs message m as $s = h(m)^d \bmod n$.

Verification: Upon receiving (m, s), Bob accepts if $h(m) = s^e \bmod n$.

The pair (s^e, s) is no longer likely to be a forgery under hash-then-RSA. It only passes verification if $h(s^e)$ matches s^e. To find a message that matches a given target signature s, Eve must invert h on output s^e. As long as h lacks special structure this requires trying out unreasonably many candidate inputs m.

Eve could also try to pick m first and then look for a signature s that is consistent with it, namely s^e that matches $h(m)$. This is equivalent to computing the d-th power of $h(m)$, which is infeasible unless Eve knows the secret key d.

Now suppose that Eve intercepts message-signature pair (m, s) with $s = h(m)^d$. The pair (m^2, s^2) is now very unlikely to be a valid forgery. It won't verify unless $(s^2)^e \equiv h(m^2)$, or equivalently $h(m)^2 = h(m)^{2de} = s^{2e} = h(m^2)$. Eve would have no reason to expect a coincidence like $h(m)^2 = h(m^2)$ to happen.

More generally, intercepting any sequence of message-signature pairs $(m, h(m)^d)$ is unlikely to help Eve. If h is modeled as a random function, the values $h(m)$ would be independent random strings, and so would be $h(m)^d$. Eve would not learn anything about other values of h by observing them. They do not help her forge any other message-signature pair $(m^*, h(m^*)^d)$.

One technical annoyance in implementing the full domain hash for RSA is that standard hash functions like SHA-3 produce bit strings as outputs. The number of possible outputs is a power of two, like 2^{256} for SHA3-256. In contrast, the number of possible outputs for hash-then-RSA is variable. It depends on the value of the modulus n, which is sampled at key generation time. Standards like RSA-PSS specify how to turn a fixed-length hash into a variable-length hash. Implementation errors in this transformation may introduce unintended structure into h leading to security vulnerabilities.

In conclusion, RSA is insecure as a signature scheme unless an "unstructured" hash h is first applied to the message. Moreover, the implementation details of h may affect security.

The full-domain hash is a general paradigm for building a signature scheme out of two ingredients. The first is a hash function h that reduces messages to be signed to unstructured fixed-length strings. The second is a *trapdoored* one-way function f that maps a candidate signature s to a message hash. To verify a message-signature pair (m, s), Bob verifies that $f(s)$ equals $h(m)$. In RSA-then-hash, $f(s)$ is the function $s^e \bmod n$.

How does Alice sign? As Alice has no control over $y = h(m)$, she must be able to pick a preimage s of y under h. Her secret key provides her precisely with this ability. A trapdoored one-way function in fact gives Alice the power to sample a uniformly random preimage s conditioned on $f(s)$ being equal to y. This sampling ability is needed to argue that the full-domain hash is secure assuming h is modeled as a random oracle. From Eve's perspective, signatures then look "independent" from their associated message hashes. Eve learns no useful information by intercepting such random-looking signatures.

In RSA-then-hash, Alice uses her private key y to "sample" s as $y^d \bmod n$. Sampling here is a purely deterministic procedure because modular exponentiation is a permutation and y determines s uniquely.

4 Public Key Infrastructure

A signature verifies a message relative to a public key. It does not, by itself, tell Bob whose public key it is. If Eve can replace Alice's public key by her own, she can sign any message with her own secret

key and claim that the signature came from Alice (see Figure 2). Verification will pass but Bob will have authenticated the wrong person.

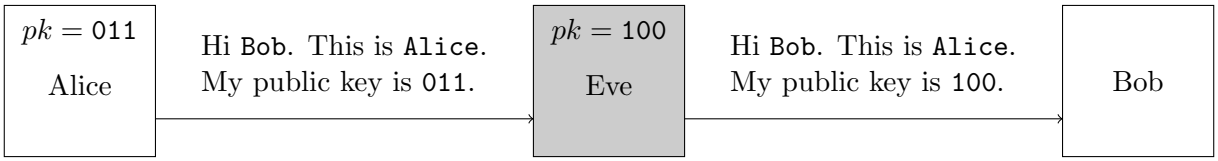

FIGURE 2: Impersonation.

A *certificate* binds an identity to a public key. A certification authority (CA) certifies this binding, signed with its own secret key. The CA, for example, may certify the message

$$m = (\texttt{Alice}, 011) = \text{“}\texttt{Alice}\text{'s public key is } 011\text{”}$$

by signing m it with its own key $\texttt{CA}.sk$. If Bob already trusts the CA's public key, he can verify the CA's signature and thereby accept the certified key as Alice's.

In web authentication the prevalent certificate format is X.509. A certificate for a website binds its domain name, e.g. `www.uottawa.ca`, to a public key. For example, `www.uottawa.ca` is certified by the CA `GTS Root R4` whose public key $\texttt{GTSRootR4}.pk$ (in hexadecimal format) is

```
00046FCD3AFE6757474C21038540C2475DBB58470F40C15C1785C61937E7D57CED
864B9B81D9D71A13A50A03F898C4C6E89EFF10598F2C2698F5E62625BB0F02FA56
```

The metadata for `uottawa.ca` binds the domain name `www.uottawa.ca` to the public key $\texttt{uottawa}.pk$

```
0004D07058F3FD1E491A4E9CDF6F09D3FCCCA58CC08B3F0FDD5F0494A33D871107
069909B9F08DAAF9A95CDC0629E8574E82272834BB6064F8B40AFDC4581582965E
```

the pair $(\texttt{www.uottawa.ca}, \texttt{uottawa}.pk)$ binding checks out as valid under verification key $\texttt{GTSRootR4}.pk$.

In addition to the public key itself, X.509 certificates contain other relevant information such as the signature scheme (X9.62 ECDSA Signature with SHA-256) as well as an expiration date (valid until 5 February 2027).

It is impractical for every user's software to contain the key of every possible issuing CA. Certificates are therefore arranged into a *chain of trust*. At the top of this chain are *root CAs* like `GTS Root R4`. There is a relatively small number of those. There are only 37 X.509 root CAs that are trusted by all four major web browsers (Mozilla, Chrome, Safari, and Edge). An end-entity certificate is signed by a sequence of intermediate CAs that eventually point to a root CA. For example, the public verification key for `andrejb.net` is certified by the chain

$$\texttt{ISRG Root G1} \to \texttt{Root YR} \to \texttt{YR2} \to \texttt{cpcontacts.andrejb.net}$$

When landing on a trusted website, the browser verifies that the requested identity matches the certificate, that the current time lies within every certificate's validity interval, that each issuer is allowed to certify the next key, and that the algorithms and key sizes are acceptable.

One practical issue that comes up in managing certificate is revocation. A certificate can expire naturally, but a compromised secret key may have to be distrusted immediately. Certificate revocation lists are one mechanism for communicating such information.

Public-key infrastructure does not eliminate trust. The software vendor trusts that root certificate authorities refrain from validating dubious entities. This trust propagates along the chain. A violation of trust can only be partially remedied by technical solutions like revocation. Legal or economic mechanisms may be necessary to complement them.

5 Combining signatures and encryption

In the private-key setting, authenticated encryption gave us communication that is both authentic and secret. Can both qualities be obtained in a public-key scheme?

In Lecture 7 we argued that the encrypt-then-authenticate combination always produces a secure authenticated encryption scheme. In contrast, a naive encrypt-then-sign public key combination has unintended vulnerabilities. Suppose Eve intercepts a signed encryption from Alice whose intended recipient is Bob. The encryption is then calculated using Bob's public key, while the signature is under Alice's secret key.

$$\begin{aligned} \text{ciphertext } c &= Enc_{\texttt{Bob}.pk}(m) \\ \text{signature } s &= Sign_{\texttt{Alice}.sk}(c) \end{aligned}$$

Eve can strip Alice's signature and resign c (without knowing what the message is) under her own secret key. She replaces s with

$$s^* = Sign_{\texttt{Eve}.sk}(c)$$

Upon receiving (c, s^*), Bob would then mistakenly think m was sent by Eve. This attack can be prevented if Alice includes her identity as part of m:

$$\begin{aligned} \text{ciphertext } c &= Enc_{\texttt{Bob}.pk}(\texttt{Alice}, m) \\ \text{signature } s &= Sign_{\texttt{Alice}.sk}(c) \end{aligned}$$

Upon reception, Bob verifies that the identity bound to the signature verification key is consistent with it.

Transport Layer Security (TLS)

In practice, (public-key) signed encryption is not as important as (private-key) authenticated encryption. The reason is that public-key algorithms consume significantly more resources than private-key ones. The usual role of public-key authentication and encryption is to establish shared secret keys for further interaction.

There is another practical advantage in using secret keys for session-specific communication like a purchase over the internet. Should the keys be compromised, only the contents of the session are affected. Subsequent sessions remain secure.

The Transport Layer Security (TLS) protocol is used to establish shared keys between a client (a web browser) and a server (a website like `www.uottawa.ca`). The steps are

1. The client Alice sends the first message of key exchange, e.g. $g^X \bmod p$ for Diffie-Hellman, for a random X. (The message includes the specification of the actual protocol used among several supported by the standard.)

2. The server Bob runs its own part in the key exchange protocol. For Diffie–Hellman, it chooses a random Y and sends $g^Y \bmod p$. Alice and Bob use the shared secret $g^{XY} \bmod p$ as a seed to a PRG to derive two secret keys ab and ba for authenticated encryption, one for each direction. Bob sends its certificate, containing its public verification key pk, encrypted under ba. It also sends a signature, under its secret signing key sk, of all the messages exchanged so far.

3. Alice computes the shared secret $g^{XY} \bmod p$ and derives the same keys ab and ba. Alice decrypts and validates Bob's certificate and verifies Bob's signature. Alice then sends Bob a MAC under ab of all the messages exchanged so far.

4. Bob verifies the MAC. If the verification passes, all further communication from Alice to Bob is authenticated and encrypted under ab, and vice versa.

The security guarantees of TLS are asymmetric. An Eve-in-the-middle can impersonate the client but not the server. This is unavoidable as the client has no public key associated with it.

The security of the key exchange guarantees the secrecy of the keys ab and ba. The server's signature validates Bob's identity. The signature in step 2 and the MAC in step 3 also ensure consistency. Without them, Bob cannot be sure that he is not talking to Alice in step 1 and to Eve in steps 2 and 3.

6 Summary

A digital signature is a publicly verifiable and transferable proof that a message was produced by the identity in possession of the matching secret key. Unlike a MAC, verification requires no shared secret. Unforgeability means that even after seeing signatures of messages of her choice, Eve cannot produce a valid signature of a fresh message.

Signatures are not the "reverse" of public-key encryption. Raw RSA signatures are forgeable because RSA preserves multiplication. Other encryption schemes are not meaningfully reversible.

Lamport's one-time signatures are built from one-way functions, i.e., functions that are easy to compute but hard to invert. They are secure for one use of the key only. Stateful chaining turns one-time signatures into a many-message scheme by authenticating the next public (verification) key. Collision-resistant hashing prevents the keys from growing exponentially in size. The security of schemes constructed in this manner is based on fairly weak assumptions, but signatures are large and expensive to compute.

RSA-PSS is a practical alternative. It hashes and randomly encodes the message before applying the RSA secret operation. Its security relies on both the hardness of factoring RSA moduli and on the unstructured nature of the hash.

Public-key infrastructure binds identities to verification keys through certificates and chains of trust. Protocols such as TLS provide both authenticity and secrecy in client-server communication. They rely on certificates, signatures, key exchange, and authenticated encryption as building blocks to accomplish those security guarantees.