CS 458 — Module 5: Internet Application Security and Privacy

1 Cryptography

- Recall Kerchhoffs' Principle/Shannon's Maxim — *assume the enemy knows the system.*
- We assume that Eve:
  - May know the algorithm
  - May know part of the plaintext
  - May have many plaintext/ciphertext pairs
  - May have access to an encryption/decryption oracle

1.1 Secret-key Encryption

- AKA symmetric encryption.
- Requires using a key to encrypt and decrypt the message.
- A theoretically perfect cryptosystem is the OTP; but this is very hard to use in practice since any key reuse ruins it, and must be truly random (pseudorandomness is not enough).
- SKES relies on mathematical problems that are hard to do in the opposite direction. This may not hold true with quantum computing for some currently-used options!
- Two main types — stream and block ciphers.
- A stream cipher takes in a keystream, XORs with the PT, and gets the CT.
- Can be very fast, and useful if you need to send a lot of data securely, but can be tricky to use correctly.
- To avoid key re-use, we may use IVs/salts to help make the key different.
- Block ciphers operate by taking part of the plaintext and spitting out the corresponding ciphertext.
- What happens when the PT is larger than one block? We have *modes of operation* for this:
  - ECB uses the same keystream for each block — but this degrades into a shitty substitution cipher!
  - Other examples are CBC, CTR, GCM, etc, that starts with an IV.
- One problem with SKES is how to share keys?

1.2 Public-key Encryption

- Allows Alice to send a secret message to Bob without a pre-arranged shared secret!
- Each user has a public key and a private key.
- These use a lot of math to allow for Alice to encrypt via Bob's public key and Bob to decrypt via his private key.
- One downside with PKES is that a key size must be *much* larger to get equivalent security to, for example, an AES SKES key.
- Another is that with larger keys, means more time. This is slow to practically use for some use cases!
- We may instead use *hybrid* cryptography. This uses PKES to encrypt smaller messages, like a key for a SKES system, then encrypt the messages via SKES!

2 Integrity

- How do we check if a message was changed in transit?
- The simplest answer is a checksum. For example, credit card numbers use checksums.
- However, simple checksums are easily faked by changing specific parts of the data; and so for our messages, this is not viable!
- So, we need a "cryptographic" checksum that Mallory could not easily forge.
- Hash functions take in an arbitrary length string and output a fixed length string, called a message digest.
- Hash functions should have 3 properties:
  1. Preimage resistance — given y, it should be one-way.
  2. Second preimage resistance — given x, it is hard to find $x' \neq x$ where $h(x) = h(x')$.
  3. Collision resistance — it is hard to find *any* two distinct x, x' where $h(x) = h(x')$.
- Collisions are always easier to find than preimages or second preimages due to the birthday paradox.
- Usually a square root of the total size possible!
- For SHA-1, for example, 2^{160} work to find a PI or SPI, but 2^{80} to brute force a collision.
- But you can't just send an unencrypted message and its hash to get integrity assurance, as then Mallory could change the message and give her own message digest!
- So, there has to still be a secure way of sending and/or storing the message digest.

3 Authentication

- Message authentication codes, or MACs, are a keyed hash function. Only those who know the secret key can generate/check the MAC.
- Typically when we combine confidentiality, we encrypt then MAC.
- Repudiation is when Bob can ensure that Alice is the one who sent the message M and it has not been modified.
- With something like a shared system, this is impossible, as Alice could claim Bob, who also has access to the hash system, made it up.
- Both repudiation and non-repudiation can be desirable based on the situation.
- For non-repudiation, we need a true digital signature, where Bob can prove to a third party that Alice sent it, and Bob cannot have forged it.
- We could use a public-key system to do so. By Alice signing via her private signature key, then Bob could verify the message with Alice's public verification key.
- We often use hybrid signatures, as signing large messages is slow. Alice can send an unsigned message and a signature on the hash of their message, for example.
- Typically, the signature key pair is long-lived, while the encryption key pair is short lived. This helps give perfect forward secrecy.
- When creating a new encryption key pair, Alice uses her signing key to sign her new encryption key and Bob uses Alice's verification key to verify this.

3.1 Key Management

- One of the hardest problems if PKES is how do we manage keys?
- Bob could find Alice's verification manually, for example (SSH).
- Or maybe get a friend to tell them, via a web of trust (PGP).
- Or trust a third party to tell them (Certificate Authority, TLS/SSL).
- A CA generates a certificate consisting of Alice's personal info and her verification key. This certificate is signed with the CA's signature key.
- So, one only needs the verification key of the root CA to verify the certificate chain.

4 Security Controls using Cryptography

- In what situations might it make sense to use cryptography as a security control?
- Remember there needs to be some separation, since any secrets need to available to legitimate users but not the adversary.
- In some situations, this makes SKES hard. If your web browser can decrypt the file containing passwords, then an adversary can as well!
- PKES would be fine if the local machine only has access to the public part of the key.
- But this would mean you can't decrypt or sign.
- We can also encrypt code. The processor would have to decrypt instructions before executing them; malware can't spread without knowing a processor's keyg.
- We can encrypt data. This provides protection if hardware containing data is stolen. It doesn't protect against legitimate users or malware or someone who can access the device while it is operational/from memory.