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.

5 Link Layer Security Controls

- Link-layer security controls provide protection at the local area network level.
- Typically LAN users are not expected to be malicious, so there's a lot less security-specific controls at the link layer.
- For example, physical security controls protecting wires.
- Or restricting their internet access for some reason.
- A potential link-layer security vulnerability is ARP spoofing.
  - ARP, or Address Resolution Protocol, is how IP addresses get mapped to MAC addresses via IPV4.
  - When the network wants to find a MAC on the network, it sends an ARP request, and the host that owns said MAC replies.
  - And it caches this; so that way the next time it needs to send something to said MAC, it already knows where to find it.
  - ARP is stateless, though, so if a host gets a reply to the ARP request, even if it didn't send a request, it'll add the reply info to its cache.

  - So an adversary can poison the cache by sending fake ARP requests containing incorrect information that wasn't requested, if the adversary is on your LAN!

- So how can we stop this?
  - A physical way is, well, don't let people connect random shit into your switches.
  - But is there a software solution?
  - We can potentially watch for when there are more ARP replies than requests.
  - Or we could create "trusted" and "untrusted" ports on a bridge/router and only allow ARP replies from trusted ports.
  - If the ARP reply comes from an untrusted port, make sure to check the details with the DHCP table at the transport layer — if this doesn't match up, then close the port!

- Another attack is MAC flooding.
  - The Content Addressable Memory (CAM) table tracks port numbers, the MAC associated with the port, and the IP if used. This is faster than storing stuff in RAM.
  - But it is a fixed size — a MAC flooding attack exploits this by sending many false ARP packets to fill up the CAM table.
  - The switch ends up confused and regresses to "hub" mode, sending traffic to everyone, including the adversary.

5.1 WiFi

- Before WiFi, compromising link-layer required premise access — this obviously changes when you add in wireless access.
- So, when WiFi was introduced, WEP was introduced to provide some security.
- WEP promised to fulfil 3 security goals:
  - Confidentiality
  - Access control
  - Data integrity
- Unfortunately, WEP was, to put it nicely, "shit". And thus, it provided absolutely none of these goals, because it was a broken piece of crap that was not designed with appropriate feedback/input.
- So how did WEP work?
  - The sender and receiver share a secret key k that is 40 or 104 bits long.
  - For the message, calculate a cheksum that does not depend on k. This is for integrity.
  - Then, pick an IV (24 bit long number), v, and generate a keystream via RC4, so we have $RC4(v, k)$.
  - Now, XOR the message M and the checksum $c(M)$ with the keystream $RC4(v, k)$ to get our ciphertext (P)!
  - Now we transmit v and the ciphertext wirelessly…
  - Now getting v and the P, if they know k, then they can also get $RC4(v, k)$. Then, XOR P and $RC4(v, k)$ to get M', the plaintext, and c', the checksum.
  - Now compare c' to $c(M')$ to verify. Done!
- So *what's wrong*?

- **Problem 0** — every WEP uses the *same* bloody secret key. So *everyone* knows this """"secret"""" keys!
- **Problem 1** — They *did* realize that they needed to use an IV to add some variety to the key. But v is only 24 bits long. It is *very* easy to break v, there are only 2^{24} combinations! So, no security.
- **Problem 2** — The checksum used in WEP is CRC-32, which is shit for this job. It does not protect against malicious errors, and CRC is already used to detect random errors, so the checksum does nothing. It's also independent of k and v, and *linear*. So it does squat for data integrity too.
- **Problem 3** — What if Mallory wants to inject a new message F into a WEP-protected network? They only need a *single* PT-CT pair that has been used and they now know the $RC4(v,k)$ of v and the keystream, allowing them to create ciphertext C' for their message by doing $[F, c(F)]$ XOR $RC4(v,k)$ and transmit v, C', the latter of which is a correct encryption of F. So it doesn't provide access control either.

  Also the PT-CT pair is given to Mallory for free as part of the auth. protocl.
- **Problem 4** — The auth. protocol involves the AP sending a challenge string to the client, and the client sends back the challenge, WEP-encrypted with shard-secret k — so this means the adversary sees the PT and the CT of the challenge! So Mallory can execute the authentication protocol.
- **Problem 5** — Mallory can even decrypt the packets! Since the AP knows k, it means the adversary can trick the AP into decrypting the packet for them.
- **Problem 6** — The WEP key is easily recoverable even if an IV is used by exploiting RC4, since RC4 using similar keys means the output keystream is weak.

- So what replaced WEP? WPA.
- How did WPA improve on WEP?
  - The client's key is a 128-bit key that gets mixed with the client's MAC address.
  - Use a 48-bit IV.
  - The key changes very frequently.
  - Use sequence counters to prevent replay.
  - Use an actual MAC (like auth. MAC, not MAC address MAC).
  - Can work on most older WEP hardware.
- This was a patch to WEP as a short-term solution, while WPA2 was being developed.
- Now, we usually use WPA2.
  - Uses AES to do the encryption rather than RC4, which WPA still used.
  - Designed from scratch with experts.
- A similar problem was mobile phones, where the original GSM implementation was broken.

6 Network-layer Security

- The communication protocol used at the network layer is IP.
- We have IPv4 and IPv6.
- IPSec is a security protocol to secure one network to another.
- IPSec gives us confidentiality (hides the real source/destination IP address and encrypts payloads) and adds checks to ensure integrity.
- IPSec also includes a way to exchange keys.

- One way of hiding traffic is “tunnelling” (not to be confused with IPSec tunnel mode). This sends a message of one protocol in another out of order of the usual protocol nesting sequence.
- A more common approach is using a VPN — basically use gateways at the origin and destination to make the physically different networks between those gateways appear to be a single network — this means that an adversary can’t read or modify traffic within the VPN! Any observer would see the origin as the gateway.
- IPSec is one way to set up a VPN. There are two modes with IPSec:
  - Transport mode (usually between endpoint and gateway):
    - Useful for connecting a device to some home network
    - Only the contents of the original IP packet are encrypted and authenticated. Source/destination IPs are exposed.
  - Tunnel mode (gateway to gateway):
    - For connecting two networks
    - The contents and the header of the original IP packet are encrypted and authenticated; and all of this is placed inside a new IP packet destined for the remote VPN gateway.

7 Transport Layer Security

- The transport layer of the stack allows host-to-host (ie: network-to-network) data movement.
- The main communication protocols at the transport layer are TCP and UDP.
- The security protocols are TLS/SSL, and the privacy mechanism is TOR.
- A segment contains of the message and the TCP/UDP protocol + port number.

7.1 SSL/TLS

- The way TLS (v1.2) works is:
  1. Client connects to the server, indicates it wants to speak TLS, and advertises what ciphersuites it knows.
  2. Server sends its certificate to clients, containing useful information like its host name, public key, signature from CA, which ciphersuite to use, etc.
  3. Clinet validates server’s certificate.
  4. If good, the client and server run a key agreement protocol to determine keys for symmetric encryption and MAC algorithms. Communiaction now proceeds using the chosen symmetric encryption and MACs.
- TLS 1.2 provides:
  - Server auth.
  - Message integrity
  - Message confidentiality
  - And optionally, client auth.
- TLS 1.3 also provides:
  - Perfect forward secrecy — so compromising the key in the future will not affect past messages!
  - A way to validate you are talking to the server you think you are talking to.
  - Many compromised/deprecated encryption schemes are not allowed.
- TLS is very successful and is important given how popular and used WiFi is.

7.2 Privacy Enhancing Technologies (PET) and TOR

- We have nodes known as Onion Routers. There are many scattered around the Internet (about 7000).
- Let's say Alice wants to connect to Bob's web server without revealing her IP address.
  1. First, she finds an entry guard node, n_1. She establishes via PKES an encrypted communication channel to n_1.
  2. She then gets n_1 to talk to another node, n_2. They share a new encrypted channel between n_1 and n_2. So, Alice knows the second new key so she can talk to n_2 via n_1, and n_2 will know about n_1 but not know who Alice is.
  3. This goes on until the last node connects to the website.
- The whole connection is called a circuit.
- The last node is the exit node.
- Typically 3 hops are required. More hops can be added but this is a speed-privacy tradeoff.
- Remember, this gives *privacy*, not security.
- So the server has no idea who Alice is at the end!
- So, sending messages with TOR:
  - Alice shares 3 (we assume 3 hops) secret keys, k_1, k_2, k_3, with nodes n_1, n_2, n_3 respectively.
  - When Alice wants to send a message M she sends $E_{k_1}(E_{k_2}(E_{k_3}(M)))$.
  - Node n_1 uses k_1 to decrypt the outer layer, n_2 uses k_2, etc., and the final destination will get M.
- When receiving messages:
  - n_3 encrypts message reply R with k_3, etc., and Alice has to decrypt all levels where she knows all keys.
- Note that the node n_1 will know that Alice is using Tor, and that her next node is n_2, but they won't know which website Alice is visiting.
- And n_3 will know some Tor user from n_2 is visiting, but not who the user is.
- The website itself only knows that it got a connection from Tor node n_3.
- Note, the connection between n_3 and the website is not necessarily encrypted — you must also use something like HTTPS to get end-to-end encryption!

7.3 Anonymity and Pseudonymity

- We have a "nymity" sliders, where we go from being identifiable to totally anonymous.
  - "Verinymity" is something that positively identifies who are you are — there is no anonymity, and you are identifiable. For example, personal identification documents like a passport.
  - "Persistent pseudonymity" is a link between who you are and who you claim to be. You are only anonymous to the extent that nobody can connect your "false" identity to your real one. For example, usernames.
  - Linkable anonymity is when patterns about your behaviour are known, but who you are isn't.
  - Unlinkable anonymity is when there isn't a link between who you claim to be, your behaviour, and who you are.

8 Application Layer Security

- At this layer this is what has been received from the network.
- We deal with 3 ways of achieving security/privacy at the application layer:
  - Remote login(ssh), email anonymity
  - PGP
  - Instant messaging, off-the-record messaging

8.1 SSH and Email Anonymity

- Two main ways to authenticate with ssh:
  - Send a password over an encrypted channel
  - Sign a random challenge with your private key
- What if we want anonymity over something non-interactive like email?
  - Anonymous remailers allow you to send email without revealing your email address!
  - Obviously this provides some issue as how do you, uh, reply?
  - Pseudonymity is useful in the context of email if you want to have a conversation (ie: mapping).
- In regards to remailers:
  - "Type 0" remailing services basically just changed the "from" address to something else. Replies to the anonymous email get backed to your real address. Note there is no encryption here.
  - "Type I" remailing services remove the central point of trust from Type 0, and messages are sent through a chain of remailers. Each step is encrpyted to avoid observers from following the messages, and remailers will delay and reorder messages. This does drop pseudonymity!
  - "nym" servers map pseudonyms to reply blocks that contain a nested encrypted chain of type I remailers, so attaching your message to the end of one of these reply blocks causes it to be sent through the chain and eventually delivered to the nym owner.
  - alt.anonymous.messages will just post their messages and public key to a forum, and when I reply to a message I encrypt via Bob's public key and upload publicly on a board. Anything Bob can decrypt is for him. Anything he can't is not.
  - "Type II" remailers use constant length messages to avoid observers watching a large file travelling through the network. There is no way to reply beyond including sender information in the body of the message.
  - "Type III" remailers allow sending and receiving anonymous emails. Messages are broken into equal sized chunks and sent encrypted node-to-node a la onion routing.

8.2 PGP

- PGP, "Pretty Good Privacy".
- Not really about anonymity, but about protecting the contents of the email messages, and providing digital signatures on email messages so you can be sure the message came from someone.
- We're going to need four things for Alice and Bob to communicate:
  1. A public encryption key
  2. A private decryption key

3. A private signature key
4. A public verification key

- To send a message M, she signs it with her signature key and encrypts the message + signature with Bob's public encryption key.
- Bob decrypts this with his private decryption key and uses Alice's verification key to check the signature.
- PGP automatically manages this, as well as allows you to sign your keyring with your own key to create a web of trust. Let's explain:
  - So, how does Alice get the authentic copy of Bob's public key, and not, say, Mallory giving the wrong key?
  - We don't have CAs here!
  - Nor could Bob just put his copy of his public key online or something, as this isn't enough to be secure! How can we trust, again, that this is actually Bob, or is it just "Bob" putting "Bob's" public key online?
  - We can use fingerprints — a crytographic hash of a key — and assuming the hash function is good we can't forge this!
  - If our hash is smaller than our public key, then it's much easier to share.
  - So, we could verify (say, over phone) that the fingerprint matches, and that's enough, rather than matching the *entire* public key.
  - But what if we can't contact the person and be sure that's them? For example, let's say I don't know Bob, I can't verify if the person I'm speaking to on the phone is going to be Bob.
  - We can sign keys! Once Alice has verified Bob, then she can use her signature key to sign Bob's key. So if you trust Alice, you trust Bob.
  - One could hold key signing parties to help build these webs of trust.
  - PGP manages *this* automatically. This is how we can manage trust in a more practical manner, without CAs!
- But what if someone breaks into Bob's computer? They could decrypt about past messages and learn the content of those messages! And they now have proof that Alice has sent those messages (no non-repudiation)!
- So Bob losing his laptop compromises Alice!
- So, PGP creates a lot of incriminating records. Alice has to be careful based on what Bob does!
- If we want messages where we don't want any record or proof of the conversation — these conversations are called ”off-the-record“ (OTR) conversations. There is no way to prove the conversation has ever taken place, both parties can just deny that they ever said anything!

8.3 OTR

- In Canada, both parties must agree to record a commercial conversation.
- In a personal conversation only one party must consent.
- We can have OTR conversations, and it is illegal to record a phone conversation if you aren't a party to the conversation.
- What about online?
- We use shared-key authentication — Alice and Bob share the same MAC key (MK).

- Only Alice and Bob know MK and they both use MK to compute the MAC.
- Bob cannot prove Alice generated the MAC, but neither can anyone else.
- This thus gives Alice a measure of deniability.
- OTR offers confidentiality, authentication, and perfect forward secrecy — all along with deniability — assuming that keys are erased.
- For example, the Signal protocol provides forward secrecy and improved deniability.
- By constantly rotating session keys we enable perfect forward secrecy,
- By using triple DH, we create deniable authenticated key exchanges, so anyone could have theoretically forged the convo. using only their public keys!