# Week 7 Readings

Jonathan Lam

10/19/21

## 1 Chapter 9: Algorithm types and modes

- Two kinds of symmetric algorithms:
  - **Block ciphers**: operate on blocks of plaintext and ciphertext
  - **Stream ciphers**: operate on streams of plaintext and ciphertext one bit or byte or word at a time; the same bit/byte/word will encrypt to a different bit/byte/word each time it is encountered
- Cryptographic **mode** combines basic cipher, some sort of feedback, and some simple operations
  - Simple operations because the security is in the underlying cipher
  - Mode should not compromise the security of the underlying cipher
- Considerations:
  - Security:
    - Patterns in the ciphertext should be concealed
    - Input to the cipher should be randomized
    - Manipulation of the plaintext in the plaintext by introducing errors in the ciphertext should be difficult
    - Encryption of more than one message with the same key should be possible
  - Efficiency:
    - Mode should not be significantly less efficient than the underlying cipher
    - The ciphertext might have to be the same size as the plaintext
  - Fault-tolerance:
    - Parallelization? Proprocessing?
    - Error recovery? Wrong/dropped/added bits?

## 1.1 Block cipher modes:

### 1.1.1 Electronic codebook mode (ECB)

- Most simple: block of plaintext encrypts into a block of ciphertext
- Easily parallelizable
- Vulnerability: **stereotyped beginnings/endings**
- Susceptible to bits added/deleted
- Need to pad to block size; **ciphertext stealing** is another alternative
- Vulnerability: **block replay**: only change certain blocks
  - A simple replay attack may be blocked by using timestamps

### 1.1.2 Cipher block chaining mode (CBC)

- Chaining adds **feedback** mechanism to block cipher
- Plaintext is XORed with previous ciphertext block before it is encrypted
- Two messages that begin the same will encrypt in the same way up to the first difference
- Thus the first block should be random data or a timestamp: the **initialization vector**
  - The IV can be transmitted in plaintext
- **Feedback** of ciphertext at encryption end and **feedforward** at the decryption end
- **Error extension**: property of taking a small ciphertext error and converting it into a large plaintext error
- **Self-recovering**: when there are only a finite number of affected blocks
- CBC recovers quickly from bit errors but not from synchronization errors
- CBC doesn't automatically detect extra blocks – plaintext should be structured to know where the message ends

- CBC allows an adversary to make a one-bit change in a specified position (but completely scrambling the previous block)
- Plaintext messages can still have patterns (over very long patterns)

### 1.1.3 Cipher-feedback mode (CFB)

- Acts like a self-synchronizing stream cipher: requires that a whole block be received before the message begins
- Data can be encrypted in units smaller than the block size
- **Why does CFB mode use encryption on both sides?**
- CFB also requires an initialization vector, which needs not be secret but needs to be unique
- **Don't understand the error propagation**

### 1.1.4 Output-feedback mode (OFB)

- Mode of running a block cipher as a synchronous stream cipher
- Similar to CFB mode, except that $n$ bits of the previous output block are moved into the right-most positions of the queue
- A single bit error in the ciphertext causes a single-bit error in the recovered plaintext
- Loss of synchronization is fatal

### 1.1.5 Counter mode (CTR)

- Uses sequence numbers as the input to the algorithm
- Solves the OFB mode problem of $n$ bit output where $n$ is less than the block length
- Counter mode have simple next-state functions and complicated outputs functions dependent on the key
- It is possible to generate the $i$th key bit (random access); may be useful for random-access decryption

## 1.2 Stream ciphers

- **Keystream generator** is simplest, outputs a stream of bits. This is XOR-ed with a stream of plaintext bits to produced the ciphertext bit.
- If the keystream generator produces the same bitstream every time it is turned on, its security will be broken; the same is true if it uses a bad source of randomness, or if the random pattern repeats
- The output of the keystream generator is a function of the key and an internal state

### 1.2.1 Self-synchronizing stream cipher

**Self-synchronizing** stream cipher: each keystream bit is a function of a fixed number of previous ciphertext bits: (**ciphertext auto key**) (i.e., the internal state depends wholly on the previous $n$ ciphertext bits); a random header of length $n$ is used for synchronization

- This has an error propagation of $n$ bits
- Vulnerable to a playback attack, if there is very frequent resynchronization and if timestamps are not used

### 1.2.2 Synchronous stream cipher

In a **synchronous stream cipher** the keystream is generated independent of the message stream (**key auto-key** (KAK))

- Encryption and decryption must be synchronized; if a bit is lost or added, the sender and receiver must re-synchronize, and must prevent parts of the keystream from being repeated
- Synchronous stream ciphers do not propagate transmission errors
- Synchronous stream ciphers do not prevent against bit toggling; if an adversary knows the plaintext, they can change bits to decrypt to whatever they want
- These are vulnerable to an **insertion attack**: an adversary can record a part of the ciphertext stream, and inject a single plaintext bit; they can get the differences to recover the original message
  - To protect against this, don't use the same keystring for different messages

  - **Does this require that the same key be used to encrypt the same message twice?**

# 2 The double ratchet algorithm

- Used by two parties to exchange encrypted messages based on a shared secret key
- New keys are derived for every double ratchet message so that earlier keys cannot be calculated from later ones
- D-H public values are attached to values, and the results of the D-H calculation are mixed into the derived keys to avoid replay attacks

## 2.1 KDF Chains

- **KDF** is a cryptographic function that takes a secret and a random KDF key and some input data and returns output data
- If the key is not secret and random, the key should still provide a secure cryptographic hash of its key and input data
- **KDF chain** refers to when some of the output from a KDF is used as an **output key** and some is used to replace the KDF key
- KDF chain has following properties:
  - **Resilience**: output keys appear random to adversary without knowledge of KDF keys, even with control over KDF inputs
  - **Forward security**: output keys from the past appear random to an adversary who learns the KDF key at some later point in time
  - **Break-in recovery**: future output keys appear random to an adversary who learns the KDF key at some point in time
- Double ratchet stores three KDF keys for three chains: **root chain**, **sending chain**, and **receiving chain**
- D-H output secrets become the inputs to the root chain; output keys from the root chain become new KDF keys for the sending and receiving chains

## 3 The Sesame algorithm: session management for asynchronous message encryption

- Sesame algorithm was desineg to managed double racket session created with X3DH key agreement, but it can work with any session-based message encryption algorithm

## 4 The X3DH key agreement protocol

- (I.e., "eXtended Triple DH")
- Provides forward secrecy and cryptograchic deniability
- Designed for asynchronous settings where one user is offline but has published some information to a server. Another user wants to use that information to send encrypted data to the first user, and also establish a shared secret key for future communication

## 5 The XEdDSA and VXEdDSA signature schemes

- XEdDSA signature scheme: describes how to create and verify EdDSA-compatible signatures using public key and private key formats initially defined for the X25519 and X448 elliptic curve DH functions
- XEdDSA enables use of a single key pair format for both elliptic curve DH and signatures; in some cases it enables using the same key pair for both algorithms
- VXEdDSA is a **verifiable random function** (VRF); successful verification of a VXEdDSA signature returns a VRF output which is guaranteed to be unique for the message and public key