# CS-501

Introduction to Malware, Threat Hunting & Offensive Capabilities Development

# Lecture 11: Symmetric Cryptography

## Applications of Cryptography

Agenda for today: overarching questions...

Operator → C2 communication: How do operators securely communicate with the C2 and vice versa?

Implant → C2 Sessions: How does the implant securely communicate to the C2 server?

Implant obfuscation: How do we frustrate reverse engineers by hiding our strings and other configuration data?

Cryptographic String Obfuscation: How do we reduce victimology down to brute force guessing?

Ransomware: How does ransomware “lock” a computer?

## Disclaimer

Cryptography is hard.

Cryptography has a rich theoretical foundation that we do not have time to cover in depth

We will make use of informal definitions, and take shortcuts when talking about types of security. To formalize it, we would need some complexity theory and number theory that we sadly do not have the time to get into right now.

The goal of this lecture is to create intuition for what symmetric cryptography is, and where to use it.

We are going to cover a semester's worth of concepts in about 1.5 hours.

For a more rigorous treatment, see CAS-CS-538

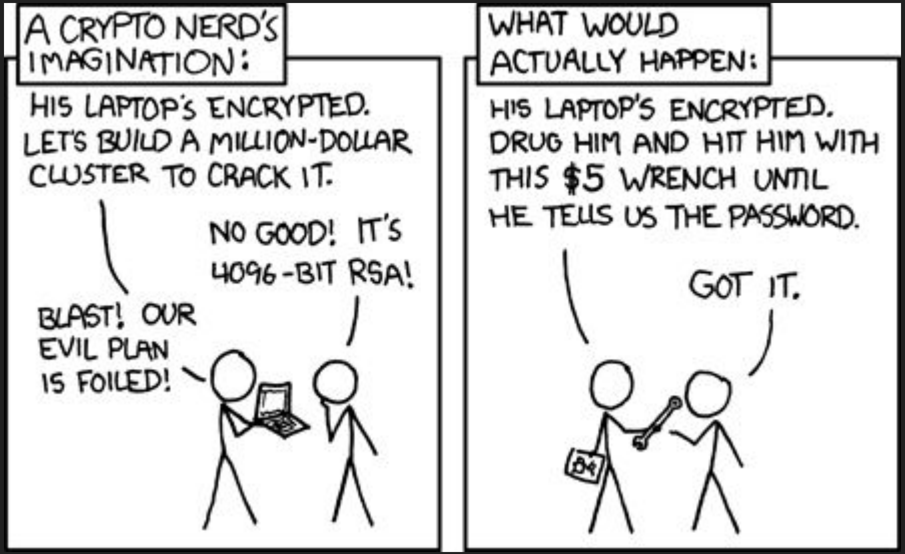
Dan Boneh's Coursera class is also fantastic and free!

## TLDR

Cryptography is very, very, very, very difficult to get right depending on who your adversary is.

Malware authors make a lot of mistakes when it comes to implementing and applying cryptographic primitives.

You probably should not “roll your own” crypto if your privacy and integrity requirements are critical.  You have been warned!


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

https://xkcd.com/538/

## Basic concepts

Basic Combinatorics: Counting functions

Basic Probability: Needle in a haystack

CRYPTOGRAPHICALLY SECURE AGAINST A COMPUTATIONALLY BOUNDED ADVERSARY
COMBINITORICS

## Basic Definitions

What exactly do we mean when we say a cryptographic algorithm is “secure”?

Breaking down “secure” into a simple case of two parties trying to communicate in an unsafe channel (Obligatory Alice and Bob)

## Basic Definitions

Plaintext: the message transmitted

Encryption: A reversible algorithm designed to provide privacy between two communicating parties who have a shared secret key.

Ciphertext: the result of encrypting a plaintext with a secret key and an encryption algorithm

Computationally unbounded adversary: an adversary with an unbounded amount of time and computational resources

Computationally bounded adversary: an adversary with limited resources

## Informal Security Definitions

Things to worry about

Privacy: can an adversary (Eve) derive “useful” information about my communication with another party?

Integrity: Can an adversary (Mallory) corrupt data without the recipient detecting it?

Authentication: How do I know I am communicating with who I think I am?

## Caesar Cipher

Start with an alphabet

Messages are composed of characters contained in the alphabet

A secret key k is a random value between 0 and len(alphabet)-1

The ciphertext is created by shifting each character of the alphabet k positions clockwise  after assigning every character in the alphabet with a position on a clock.

The plaintext is recovered by shifting each character counter clockwise k positions

## Security of the Caesar Cipher

When the length of the message > 1: it is not great

We can check all possible keys if the size of the alphabet is reasonable

I.e., it is vulnerable to a brute force search

## One Time Pad

Message: bytes of length n

Key: random bytes of length n

Encrypt: compute ciphertext= message xor key

Decrypt: compute plaintext = ciphertext xor key

## Limitations of the One Time Pad

- As the name suggests, you should only use the random byte stream once
- Encrypting  large volumes of data  requires a large pool of entropy
- Ciphertexts are “malleable”

## Note:

Remember, any data that you leave in a binary can be recovered by a skilled reverse engineer

In other words, you should assume that they get leaked!

## Symmetric Cryptography (simple case)

Cryptographic protocols between two parties that have agreed on a shared secret.

## Perfect Secrecy

- No amount of computational power can give you odds better than just guessing
- Only the One-time pad has this property

## Security: Computationally Bounded Adversary

The “advantage” an adversary gets in predicting your plaintext after seeing it’s ciphertext is exponentially small (negligible)

A more rigorous treatment of this topic is beyond the scope of our class, but the idea here is we want to create an algorithm that without knowledge of the secret key, the adversary has a negligible change of recovering the plaintext.

## Security Vs Obscurity

As we will see, it is usually easy to identify implementation of common block ciphers and stream ciphers. In the context of extracting configuration data from an implant, is it better to use a robust, easy to identify cryptographic primitive, or an ad hoc weaker primitive?

## Threat model: Malicious OS

From the perspective of the malware author, when leveraging cryptography we must assume that the OS the implant is executing on is malicious.

## Encryption vs Obfuscation vs Cryptographic Obfuscation

## Stream Ciphers

- A cryptographic algorithm that acts a lot like the One time pad, with the exception that the stream of pseudo-random bytes are generated by using a cryptographically secure Pseudo Random Number Generator (csPRNG)
- The algorithm can be safely used synchrony

## Common Stream Ciphers

RC4

Salsa20

ChaCha20

## Block Ciphers

Encrypt data in chunks called blocks

If the size of the data is not a multiple of  the block size, it needs to be “padded”

If the size of the data is larger than the block size, the algorithm needs a “mode of operation”

Those two requirements have created a pretty massive headache for cryptographers and are hard to get right.

## Common Block Ciphers

Rijndael/AES

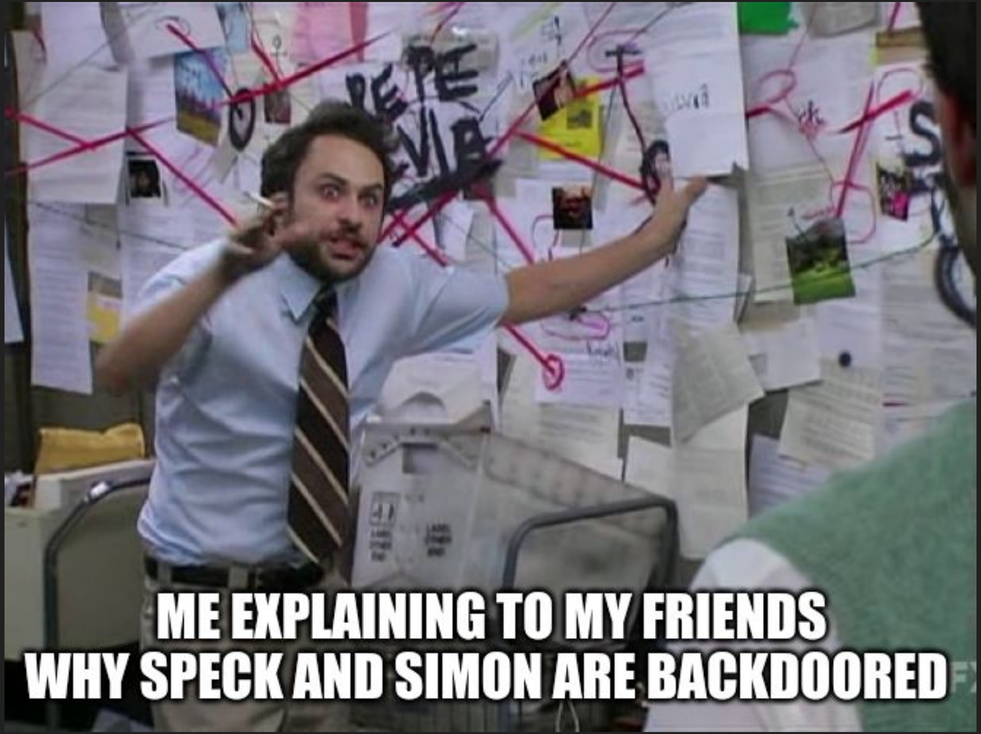
Blowfish

TEA/xTEA

SIMON/SPECK (Sus :-)

DES  3DES

RC5


ME EXPLAINING TO MY FRIENDS
WHY SPECK AND SIMON ARE BACKDOORED

## Common Mistakes

Messing up the padding. There are a lot of ways to do this.

Using ECB mode for a block cipher

Reusing nonces in CBC mode

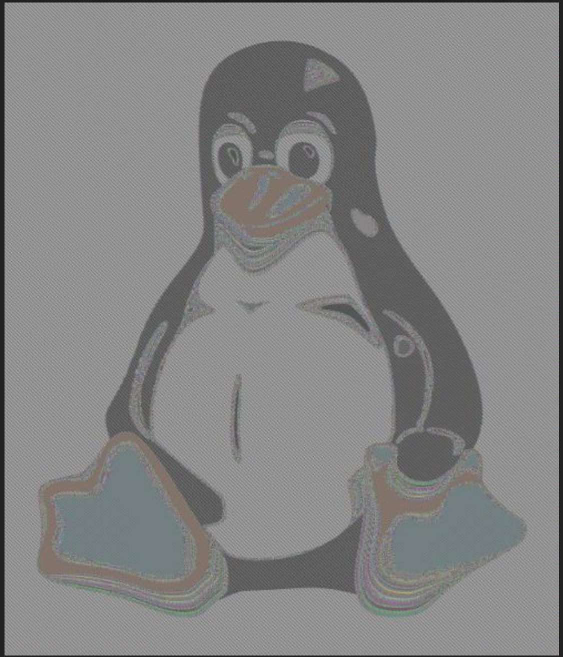
Using weak algorithms (rot ciphers, substitution ciphers)

Inadequate authentication to the server

Inadequate server authentication to the client

Failure to use MACs

Using weak hash functions



## Practical Attacks on Crypto

This class will focus on attacking crypto used by malware from the perspective of a reverse engineer.

In particular, we will look at how we can recover keys, replay messages, inject our own data, and  decrypt traffic.

## Recovering the Key From the Malware

Statically: hard if they hide it

Dynamically: usually easy. Find the function that encrypts/decrypts and just set a breakpoint. How do you find that function? Set breakpoints on common WinCrypt API functions, look for constants, trace your way backwards after network communication...etc

## Example: RC4

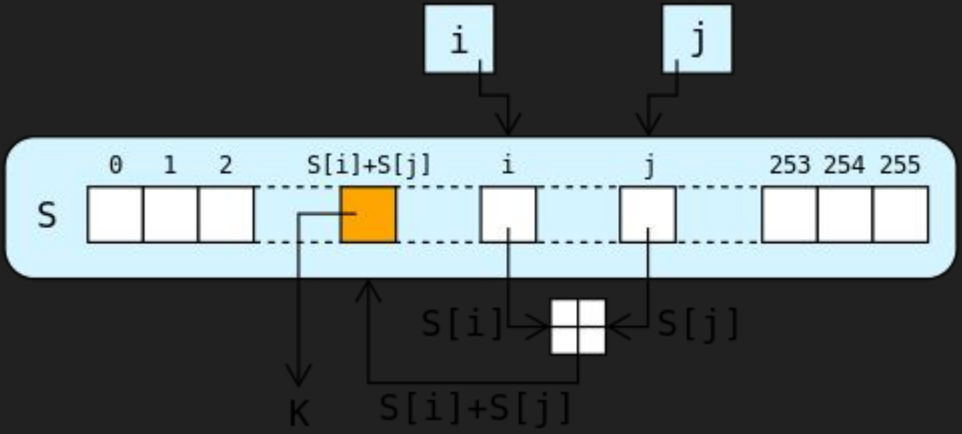
RC4 is very likely broken. It should not be used in any production environment and was removed from TLS. So why are we talking about it?

Well, RC4 is an incredibly simple cipher to implement, and a fan favorite of malware authors.

# RC4

- Key scheduling algorithm
- Stream generation

We will look into this more next week



https://github.com/kbsec/misc_crypto/blob/master/rc4/rc4.py

## Identifying Common Crypto

- Constants
- Common Round Functions
- Example, using a ghidra plugin to identify sha256 in crackme 4.

https://github.com/TorgoTorgo/ghidra-findcrypt

## Integrity Vs Privacy

- Privacy: nobody can read what your messages are
- Integrity: if anyone modifies your data in transit, you will detect it

## Message Authentication Codes

- Symmetric protocol to detect  data corruption/tampering
- Simplest example of this is HMAC (Hash MAC)

HMAC: Fun fact, invented by a BU professor: Ran Canetti

## What should you use for C2 Sessions?

Generally Believed to be secure

AES-GCM

CHACHA20-Poly1305

Remember, all of this is moot if they simple pull the symmetric key out of your binary or dump it from memory. You should assume your session keys will eventually get leaked the same way you should plan for your implant eventually getting caught!

## Solid Libraries

OpenSSL, LibreSSL, python Cryptography, Rust Crypto, LibSodium, Monocypher, NACL

For your implant, I would recommend using a combination if Windows Crypto API functions along with a statically linked cryptography library. OpenSSL is very large, as is LibreSSL. An experimental library that is targeted for embedded devices is LibHydrogen and WolfCrypt are good options.

I have also personally made use of https://github.com/jedisct1/libhydrogen/tree/v0 (old branch)

Note that the cryptography you use is something that can be used to identify you

## What is missing?

Problem: If your key gets leaked, then all previous communications are also easily decrypted.

Problem: How do we get keys distributed?

Problem: How do we authenticate to the server? How does the server authenticate to us?

## Opsec Considerations

- Using custom crypto can identify you
- Your implant is executing on a malicious OS. This makes privacy very difficult.
- Your goal should be to blind the defenders until you have achieved your goal
- Cryptography can hide your data, but it is harder to hide the fact that you are using cryptography!

## Cryptographically Secure Hash Functions

Preimage Resistance

Second Preimage Resistance

Arbitrary Collision Resistance

## Using Hash Functions Effectively

- Keying implants to the environment
- Deriving symmetric keys  from environmental values and
- Checksum/tamper detection
- API Hashing

## Where to learn more?

Dan Boneh is a pioneer in the field of cryptography, and has an incredible, free textbook and course.

https://crypto.stanford.edu/~dabo/

## Open Discussion: Ransomware