Cryptography

Many Time Pad

RSA

Man in The Middle Attack

Meet in the Middle Attack

- Join the Slack: siginthqv2.slack.com
- Join us for CTF Saturdays (first one next weekend - CodeGate)
- Workshop on hacking 👩💻 ₿lockchains with NetCraft: 10th February 🛰
- Social will be announced soon 🍻

DISCLAIMER

- All of the material in this school can be used for good (testing, research, educating), but also for bad
- We use our skills and knowledge responsibly and ethically
- We recommend you do the same
- We are not responsible for anything you do as a result of these lessons

Reminder of the One Time Pad

Suppose that a party A has a message M that is n-bits long, and wants to send it to party B. Also suppose that A and B have a key K that is n-bits long.

A sends: $C = M \oplus K$

B decrypts: $M = C \oplus K$

The key K should be generated randomly and every message must use a different key. This ensures that the ciphertexts are not susceptible to cryptanalysis as every plaintext is equally possible and there is no way of knowing which is correct.

Any eavesdropper cannot read the message!

Many Time Pad

A one-time key is when only one key is used per message.

A many-time key is when multiple messages may be encrypted using the same key.

When the same key is used for more than one document we can exploit this.

Let’s put Stream Ciphers into the mix

Stream ciphers are ciphers that work as follows:

They produce a keystream from an original key (usually just the key), and they XOR the keystream with the plaintext.

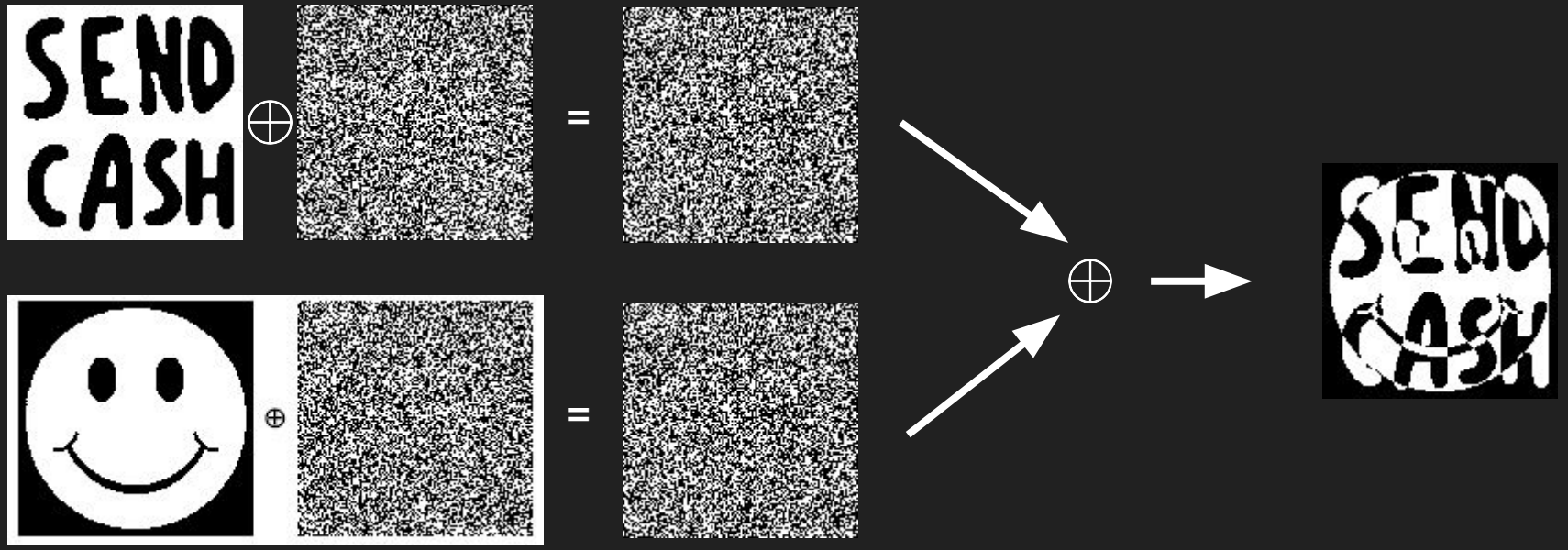
Key K → (Magic Box) → Keystream KS of “unlimited” length

Same K results in same keystream! If we use K twice, then the KS would be the same, and an attacker can deduce one plaintext if they know the other.

$C1 \oplus C2 = (M1 \oplus KS) \oplus (M2 \oplus KS) = M1 \oplus M2$

Solution: Use randomness to produce the keystream for every encryption.

Example of MTP: M1⊕M2



Another example of this attack on MTP

If we are provided many ciphertexts produced with the same key, if we XOR pairs of them, we get XORs of plaintexts.

Even if we do not know any of the plaintexts, we can extract information.

Suppose that the messages are English text. Then the 5 character string “ the ” will be very common. For a given XOR of 2 messages, we can XOR that string in all positions. What remains should make sense in some positions. We can try it with other XOR pairs to verify, and deduce parts of the original texts.

Also, if we XOR a punctuation character (usually space) with a letter, bit 6 of the result will be 1. That can be useful to find the places that spaces occur.

Attacking a many time pad

https://github.com/CameronLonsdale/MTP

Symmetric and Asymmetric Crypto

So far we have seen the notion of Symmetric Crypto. That is, two (or more) parties share a key, and they encrypt their messages using that key, and they decrypt using that key as well.

Asymmetric crypto is different. Each party has its own private and public key.

- From a private key, a public key can be extracted. The opposite does not hold.
- Everyone can learn the public key, and encrypt with that. Only the owner of the private key can decrypt.
- The owner of the private key can encrypt, and everyone can decrypt with the public key. Therefore, that can be used as a way to “sign” messages.

Numbers...

So far we have talked about strings of bits, and how to encrypt them (for example, we XOR them with a key of equal length).

Now we will talk about encrypting and decrypting numbers. But how, using numbers, we can encrypt/decrypt files or other things?

Simple: We use the binary encoding of what we want to encrypt, which is a number.

What is RSA?

RSA is a form of public-key/ asymmetric cryptography

RSA is used for secure data transmission

There is a public key for encryption and a private key for decryption.

The algorithm is based off of prime number factorisation

The algorithm itself is quite slow

If (d,N) is the decryption (private) key and (e,N) is the encryption (public) key and P is the plaintext, then the ciphertext C is calculated by $C = P^e \bmod N$

This can be recovered $P = C^d \bmod N$

RSA Algorithm

1. Choose 2 primes p and q (large numbers randomly generated), p!=q
2. Compute the product p*q, Let N = p*q
3. Calculate the totient φ(N) = (p-1)(q-1)
4. Then we have 2 equations:
    a. EncryptPrime * DecryptPrime = 1 mod φ(N)
    b. (φ(N) * AnyInteger) + 1 = 1 mod φ(N)
5. Merging these equations: (RHS has exactly 2 prime factors)
    a. EncryptPrime * DecryptPrime = (φ(N) * AnyInteger) + 1
6. Choose a value that is 1 mod φ(N) value with exactly 2 prime factors
7. Let e = EncryptPrime and d = DecryptPrime, C = ciphertext and P = plaintext
8. Then to encrypt C = P^e mod N
9. To decrypt P = C^d mod N

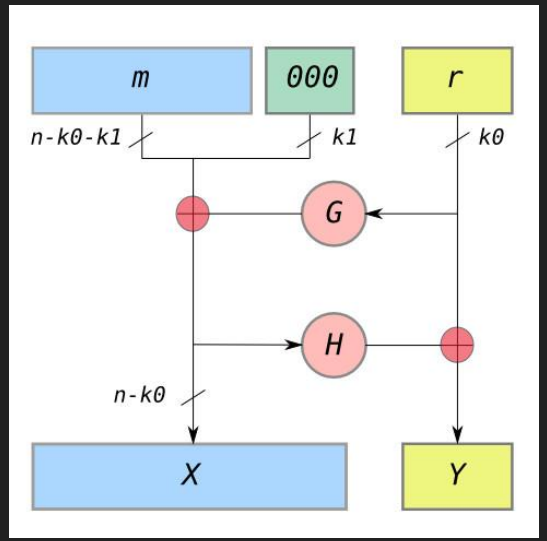
RSA attacks

RSA should be used very carefully, since there are a lot of small details that can result in breaking the security.

https://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf

Proper usage of RSA: use OAEP before encrypting

That will provide randomness and protect against some types of messages that are in dangerous ranges (eg too small).



Simple attack on RSA

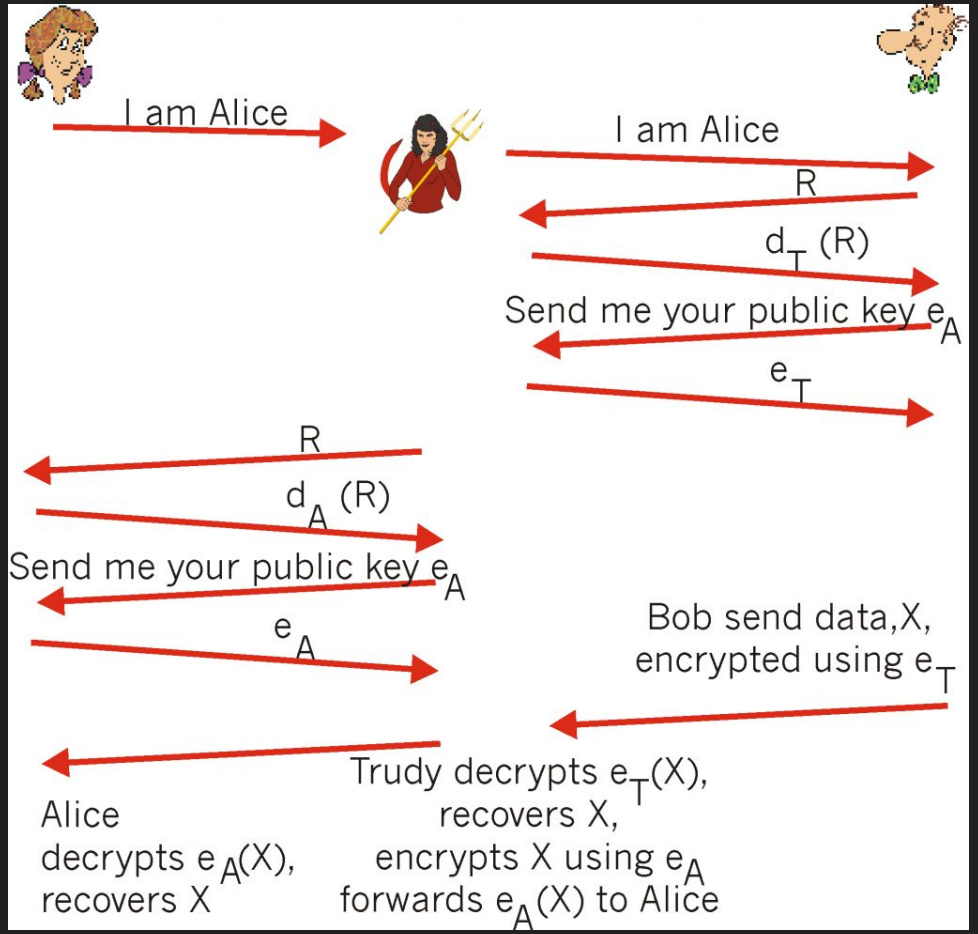
Small M and e: What if e is small (e.g. 3) and M is also small? Then it might be the case that $M^e < N$, therefore $C = M^3 \bmod N = M^3$. In that case M can be extracted trivially.

Fooling a signature: We want Bob, who has public key (N,e) to sign a message M. We give him $M' = r^e M \bmod N$,where r is a random number . Bob signs the innocent-looking M' , and we get $S' = (M')^d \bmod N$. If we divide by r, we get $S = S'/r \bmod N$, which is the signature for M. To verify:

$S^e = (S')^e / r^e = (M')^{ed} / r^e = M' / r^e = M \pmod N$

Man in the Middle Attack

This is a general attack, that can be used against symmetric or asymmetric crypto systems.



Meet in the Middle attack

General method for bruteforcing an unknown string.

Consider encryption algorithm, and unknown keys K1,K2 (each of length n)

C = Enc(K2, Enc(K1, P))

P = Dec(K1, Dec(K2, C))

Assume that we know P,C and we want to find the keys.

Normal bruteforce complexity: $2^{(n+n)}$ operations.

Meet in the Middle attack

C = Enc(K2, Enc(K1, P))

P = Dec(K1, Dec(K2, C))

K1 and K2 are independent. If we can store the possible results of Enc(K1,P), we can try all the possible Dec(K2,C) and see if they match with any of the stored values.

Complexity: $2^{(n+1)}$ operations (+ the search overhead for every K2) .
Also, 2^{n} storage space is needed.

Challenges

https://school.sigint.mx

Links

https://en.wikipedia.org/wiki/RSA_(cryptosystem)

http://doctrina.org/How-RSA-Works-With-Examples.html

https://en.wikipedia.org/wiki/Meet-in-the-middle_attack

https://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf