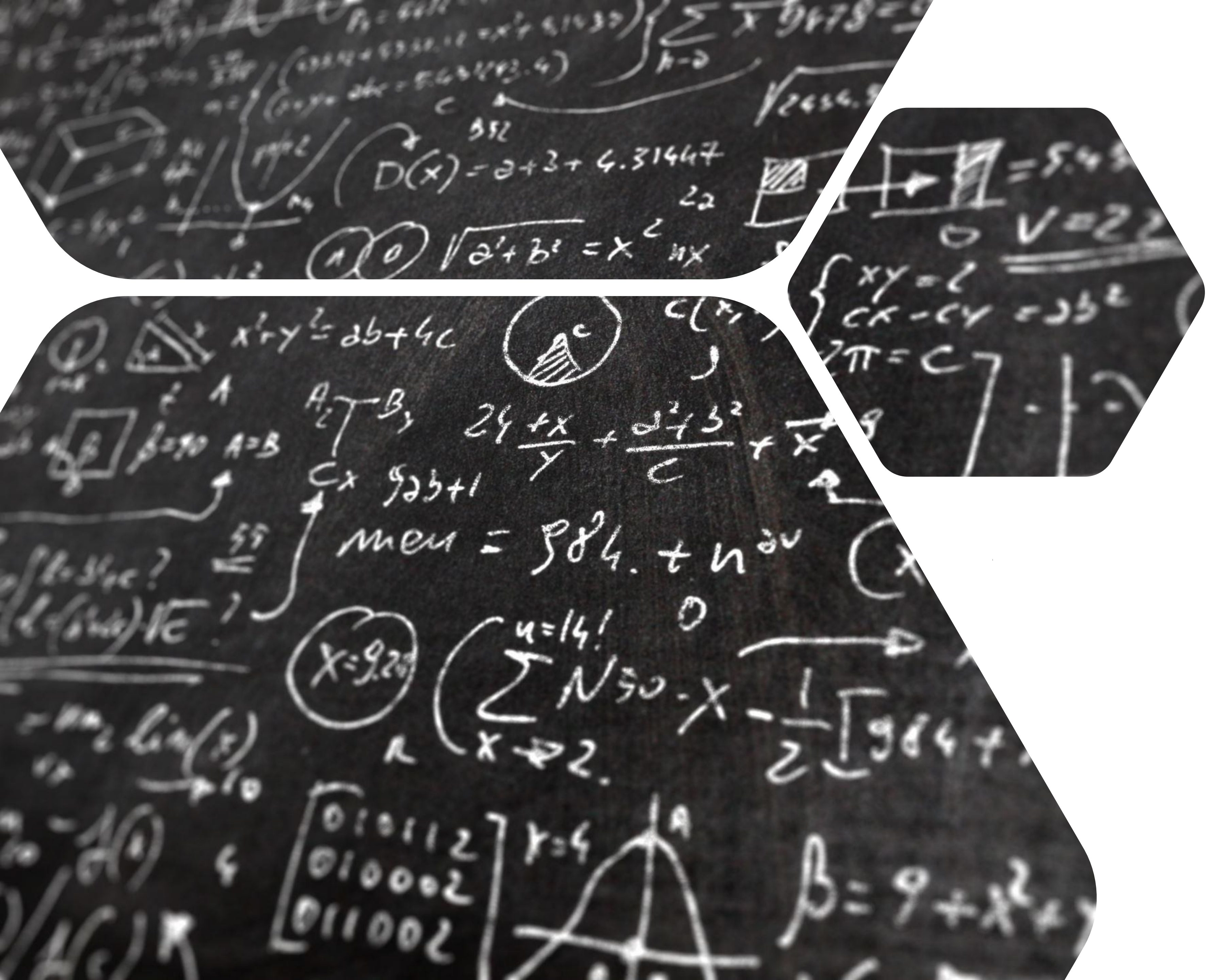


# Signatures in Cryptography

## Cryptography: *κρυπτός* = hidden secret and *γράφειν* = to write

- Cryptography exists decades before cryptocurrencies existed
- First cipher: the scytale
- First steganography app: tattoo on shaved heads
- Second world war: Enigma cipher broke helped allies won the war
- Modern cryptography: ~1970-now

## Prime in cryptography

- Encryption protects **privacy/confidentiality** of information (plaintext)
  - Only the owner of the **secret key** can decrypt (owners of secret keys are trusted)
  - **Symmetric:** Encryption and Decryption keys are the same and should be kept **secret**, e.g: AES, (X)ChaCha2020
  - **Assymetric**: Encryption key is **publicly** available to everyone and knowledge thereof **does not harm** security of the encryption scheme. **Secret decryption key** should be kept **secret** (RSA Encryption, Elgamal, Paillier)
  - **In practise**:
    - Symmetric key cryptography is very fast compared with Assymetric.
    - But how parties in symmetric cryptography exchange their secret key in public channels?
    - Hybrid setup(e.g: TLS): Use Assymetric cryptography to agree on a secret key, use that secret key to encrypt with symmetric cryptography all the entire communication

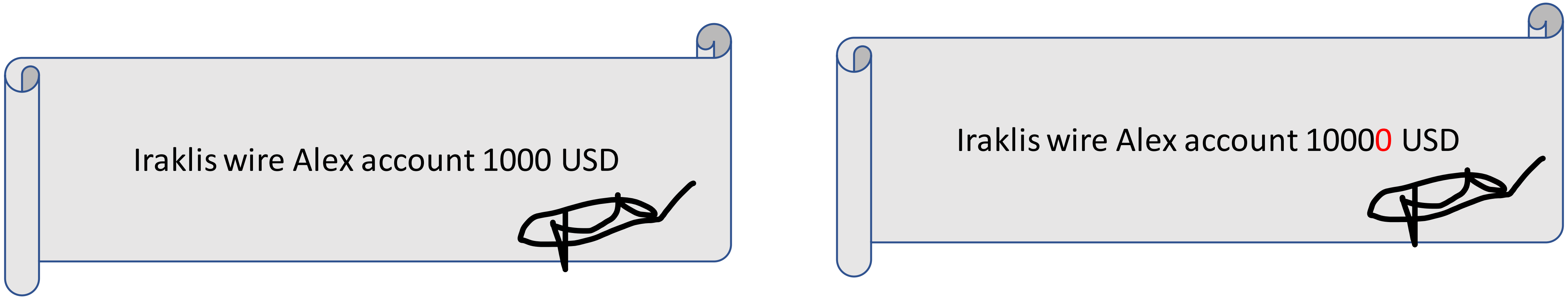
# What does it mean encryption X is secure?

- We have to define our security model:
  - What does the adversary know (attack model) ?
  - What the adversary is trying to break in a system, or what we consider as a successfull attack

| Attack model/Attack | Ciphertext only | Known plaintext | Chosen plaintext | Chosen ciphertext |
|---|---|---|---|---|
| Key extraction | Insecure | | | |
| Plaintext decipher | | | | |
| Some information about plaintext (e.g: 1st bit) | | | | |
| Distinguish ciphertexts | | | | Secure |

# Signatures in the physical world

- Problem: How to bind a physical person with a document



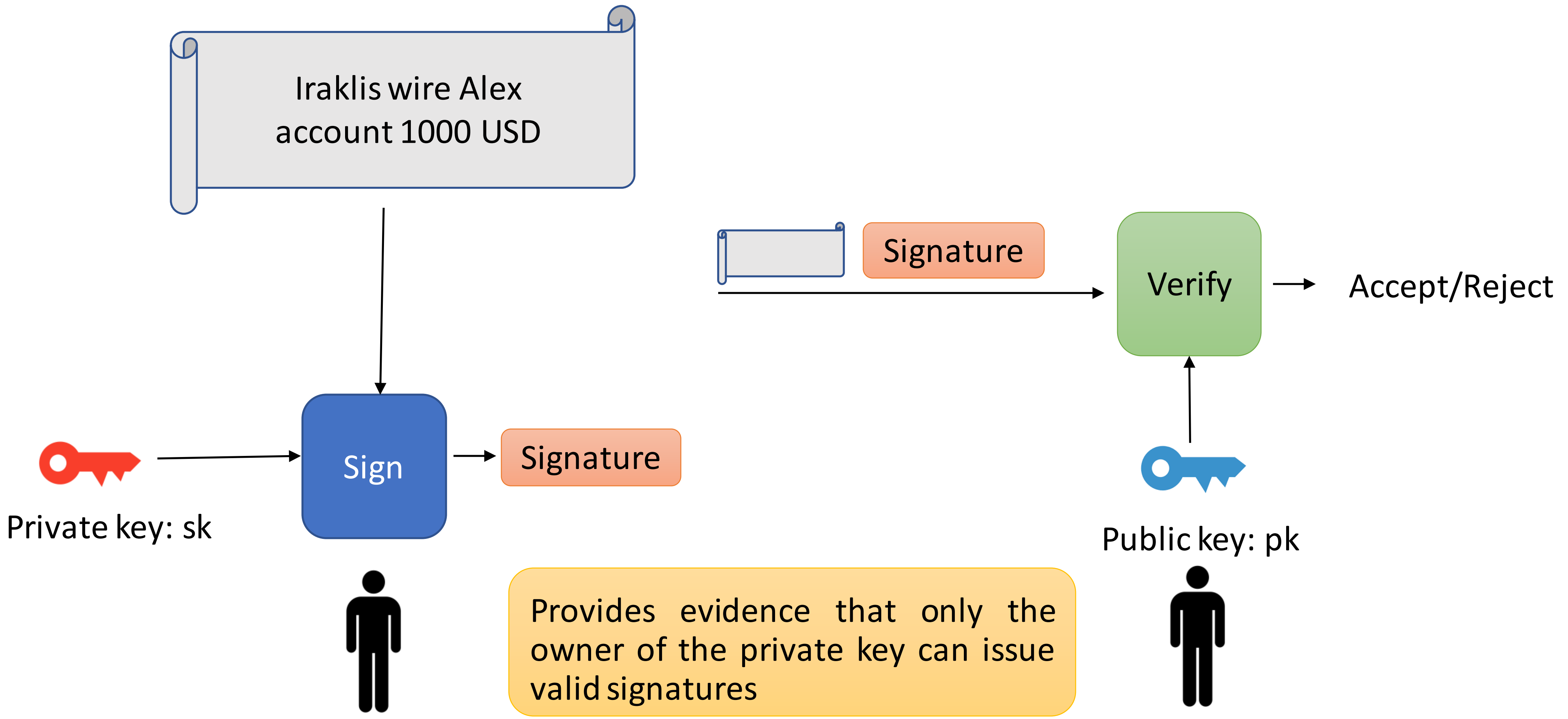
Iraklis wire Alex account 1000 USD

Iraklis wire Alex account 10000 USD

Can be easily forged

# Digital signatures (no privacy on the message)

- How to prove ownership/integrity on a bytestream



Provides evidence that only the owner of the private key can issue valid signatures

## Signatures Security

- Adversary cannot **tamper/change** not even a single bit from the message by having access to the **message**, its **signature** and the **public key**
- Signatures **SHOULD NOT** be used if **confidentiality/privacy** of message needs to be protected.
- Signatures **are not** designed **to protect the privacy** of the message: It is ok the adversary to know the message in plaintext and still have secure signatures
- Signatures protect **integrity** of the message: Receiver is getting assurances that the message has not and cannot been changed by intermediate entities without knowledge of the secret signing key after it has been signed by the sender.

## Signatures Security (contd.)

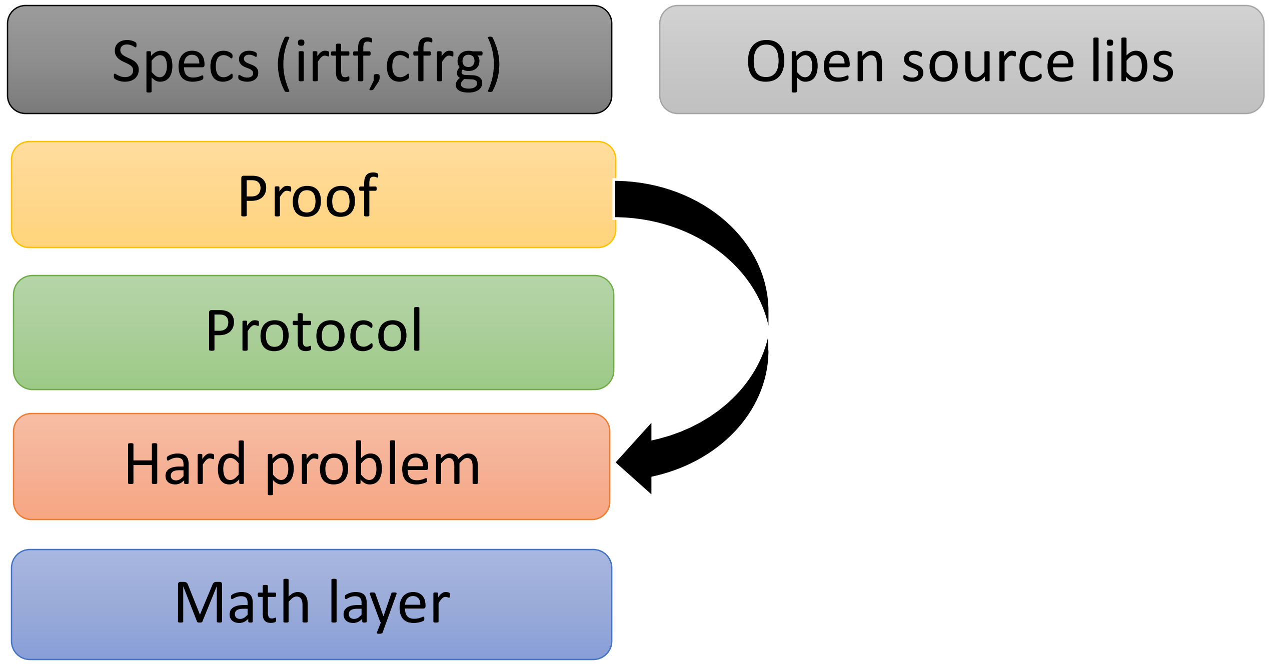
| Attack model/Attack | Key only | Known message | Adaptive chosen message |
|---|---|---|---|
| Total break | **Insecure** | | |
| Universal forgery | | | |
| Selective forgery | | | |
| Existential forgery | | | **Secure** |

# A Real World example

- Director of engineering: *We need to install that library from xyz.com*
- Engineer: *Ok, let's download it from xyz.com*
- Security engineer: *How we know that the software is reliably issued from xyz.com?*
- Xyz.com: *All packages are signed with our private key, for which we have a certificate from PKI X.*
- Security engineer: *Ok their public key is signed with the PKI X private key, certifying that the claimed pk of xyz.com is owned by them.*

CAs have been compromised many times, e.g: Diginotar, Monpass

# Cryptographic primitive Layers



## Prime in basic number theory

- Groups
- Hard problems
  - DL
- EC in cryptography

## Group

- A Group **G** is a finite set of elements with a binary operation op (e.g: +, *) such that:

- [**closure**] for all **a,b** in **G** we have **a op b** is also in **G**
- there exists an **identity** **id** such that for all **x** in **G** we have **x op id = x**
- for every x there exists an **inverse** thereof **x.inv**, such that **x op x.inv = id** (0 for +, 1 for *)
- [**associativity**] for all **a,b,c** in **G** we have (a+b)+c =a+(b+c)
- [**commutativity**] for all **a,b** in **G** we have a+b =b+a (if this holds, the group is called abelian)

## Examples

- **R** (reals) is not a group with multiplication.
- **R** \ {0} is a group with multiplication.
- **Z** (integers):
  - is a group under addition (identity element: 0),
  - is not a group under multiplication.
- $\mathbf{Z_N}$ = {0,...,N-1} (integers modulo **N**) is a group under addition modulo **N** (identity element: 0)
- If **p** is a prime then $\mathbf{Z_p^* = \{1,\ldots, p\text{-}1\}}$ is a group under multiplication modulo **p**

# Hard problems

- Discrete logarithm (in certain groups):
  - Given $g^{x} = y$ to find **x it is believed hard** ($x = \log_g y$)
  - in $Z_n$ it is easy because $g^{x} = g \cdot x \bmod n$
  - In $Z_p^*$(where p is prime) it is believed to be hard.

Extremely useful group since all elements can produce all other elements! That uniform structure makes **DL problem** hard in real world cryptographic protocols

# Elliptic curves

- An **EC** is the set of points **{x,y} \in R** that validate the EC equation
- Together the point **O** of infinity
- (EC,+) is a group:
  - Closed under addition : (x1,y1)+(x2,y2) = (x3,y3) also in **EC**
  - Identity element = **O**
  - Inverse (x,y) = (x,-y)
  - Associativity holds

**Weierstras**

- $y^2=x^3+ax+b$

**Mongomery**

- $By^2=x^3+Ax^2+x$

**Edwards**

- $x^2+y^2 = 1+ dx^2y^2$

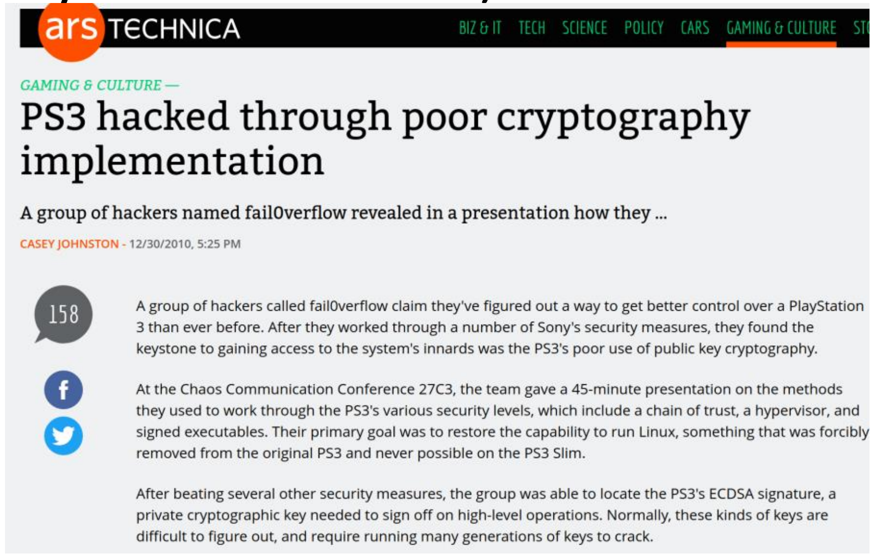
# Elliptic Curve Cryptography

- In some EC there is 1-1 map with a finite field
- Instead of working in **R** in cryptography we use EC under a field (**$Z_p$**)
- Same security guarantees with smaller parameters!

- EC(p,c,G,n,N,h) parameters:
  - The **prime p** that specifies the size of the finite field.
  - The **coefficients** c of the elliptic curve equation.
  - The **base point G** that generates EC points
  - The **order n** of the subgroup
  - The **order N** of the EC.
  - The **cofactor h** of the subgroup: h = N/n (extra care)

## ECDSA

- Descendant of DSA(FIPS 186-4, ANSi X9.62, IEEE P 1363)
- Not patended (that is why probably has been chosen by bitcoin in 2008)
- Weak security proof
- Implementation can easily go wrong
- PK can be recovered from signature
- First versions malleable



GAMING & CULTURE —

PS3 hacked through poor cryptography implementation

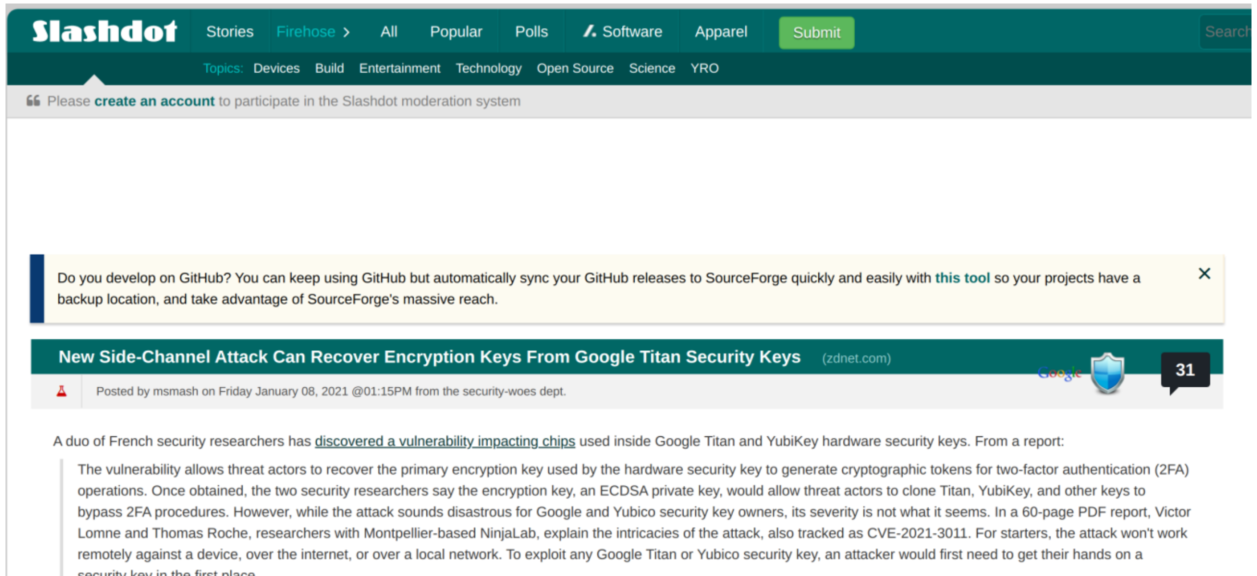
A group of hackers named fail0verflow revealed in a presentation how they ...

CASEY JOHNSTON - 12/30/2010, 5:25 PM

A group of hackers called fail0verflow claim they've figured out a way to get better control over a PlayStation 3 than ever before. After they worked through a number of Sony's security measures, they found the keystone to gaining access to the system's innards was the PS3's poor use of public key cryptography.

At the Chaos Communication Conference 27C3, the team gave a 45-minute presentation on the methods they used to work through the PS3's various security levels, which include a chain of trust, a hypervisor, and signed executables. Their primary goal was to restore the capability to run Linux, something that was forcibly removed from the original PS3 and never possible on the PS3 Slim.

After beating several other security measures, the group was able to locate the PS3's ECDSA signature, a private cryptographic key needed to sign off on high-level operations. Normally, these kinds of keys are difficult to figure out, and require running many generations of keys to crack.



Please **create an account** to participate in the Slashdot moderation system

Do you develop on GitHub? You can keep using GitHub but automatically sync your GitHub releases to SourceForge quickly and easily with this tool so your projects have a backup location, and take advantage of SourceForge's massive reach.

**New Side-Channel Attack Can Recover Encryption Keys From Google Titan Security Keys** (zdnet.com)

Posted by msmash on Friday January 08, 2021 @01:15PM from the security-woes dept.

A duo of French security researchers has discovered a vulnerability impacting chips used inside Google Titan and YubiKey hardware security keys. From a report:

> The vulnerability allows threat actors to recover the primary encryption key used by the hardware security key to generate cryptographic tokens for two-factor authentication (2FA) operations. Once obtained, the two security researchers say the encryption key, an ECDSA private key, would allow threat actors to clone Titan, YubiKey, and other keys to bypass 2FA procedures. However, while the attack sounds disastrous for Google and Yubico security key owners, its severity is not what it seems. In a 60-page PDF report, Victor Lomne and Thomas Roche, researchers with Montpellier-based NinjaLab, explain the intricacies of the attack, also tracked as CVE-2021-3011. For starters, the attack won't work remotely against a device, over the internet, or over a local network. To exploit any Google Titan or Yubico security key, an attacker would first need to get their hands on a security key in the first place.

# ECDSA Description (informal)

- KGen()->sk,pk
  - sk a random scalar
  - pk =sk *G, where G is the base point

Most of the attacks come from reusing k in code

- Sign(sk,m)
  - Pick uniform at random k (integer)
  - Compute R = k * G (ec point)
  - Let R=(rx,ry) (ec point)
  - s = (H(m) + sk *rx)/k (integer)
  - Output (rx,s)

- Verify(σ,pk,m):
  - a=H(m)/s, b=r/s (integers)
  - u = G * a + pk * b (ec point)
  - u=(u_x,u_y) (ec point)
  - rx' = u_x

- U=G*a+pk*b=G*H(m)/s + (sk*G*r)/s

  = ((H(m)+sk*rx)/s)*G = k*G = R

## Schnorr Signature

- Simpler (linear->multisig optimality)
- Formal proofs from standard assumptions. They were under patent law of Claus P. Schnorr till 2008 😠
- Non-malleable
- Lack of public key recovery

# Schnorr Signature(informal, no group notation)

- KGen()->sk,pk
  - sk a random integer
  - pk =sk *G, where G is the base point
- Sign(sk,m)
  - Pick uniform at random k (integer)
  - Compute R = k * G (ec point)
  - Let R=(rx,ry) (ec point)
  - s = k + Hash(rx||pk||m)*sk (integer)
  - Output (rx,s)

- Verify(σ,pkσ,pk):
  - If s*G == R + Hash(rx||pk||m)*pk accept, otherwise reject.

## EdDSA (Ed25519)

- Dual standarization efforts: NIST-FIPS and IETF
- It avoids side channel attacks
- No need for nonces
- Faster compared with Schnorr and ECDSA
- TLS 1.3, SSH, Tor, GnuPGP, Signal
- Corda, Tezos, Stellar, Libra

# EdDSA(informal)

- KGen()->sk,pk
  - sk a random scalar
  - pk =sk *G, where G is the base point

- Sign(sk,m)
  - $r$ = hash(hash($sk$) + $m$) (integer)
  - R = r * G (ec point)
  - $h$ = hash($R$ + $pk$+ $m$) (integer)
  - $s$ = ($r$ + $h$ * $sk$)   (integer)
  - σ = ( $R$, $s$ )

- Verify(σ,pkσ,pk):
  - $h$ = hash($R$ + $pk$+ $m$)
  - $P1$ = $s$ * G
  - $P2$ = $R$ + $h$ * $pk$

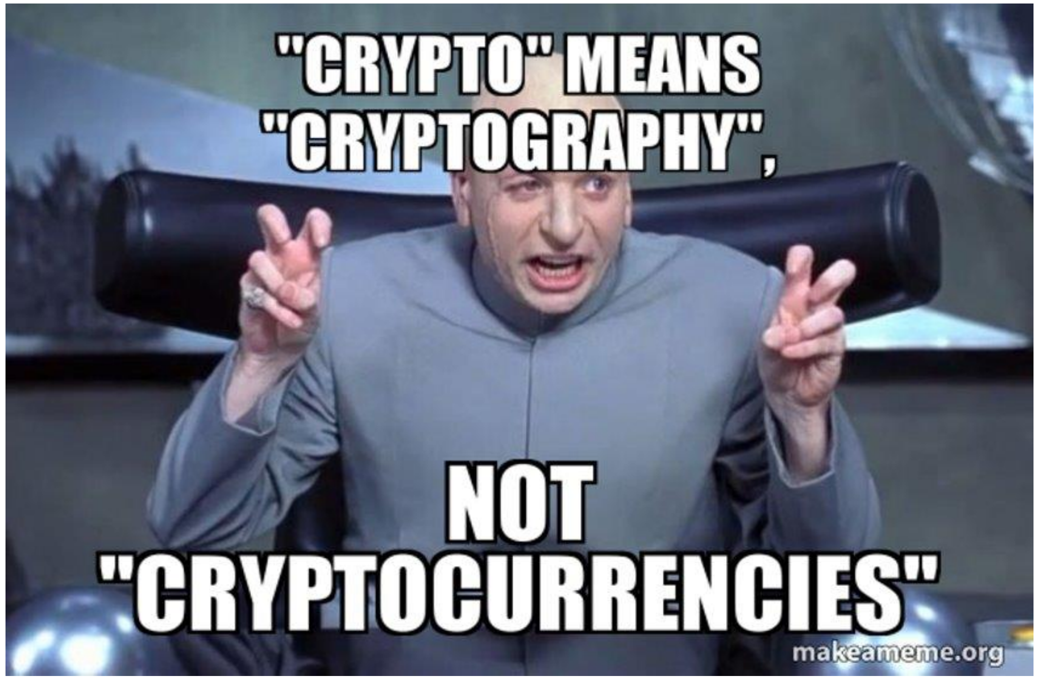
# Curves in signatures

- ECDSA: secp curves and other in Weierstras form
- Schnorr: ristretto25519, *secp curves*
- EdDSA: ristretto25519, Edwards: Curve25519

- Edward curves outperform ECDSA common curves:
  - Faster
  - Easier to program (no extra cases in addition law)
  - Safer to avoid timing attacks

## Final Takeaways

- Security of the cryptographic scheme relies on the entire system and in not in an isolated "paper" world
- In the end no matter of what cryptographic secure signature is used, the scheme is as secure as the secrecy of the secret signing key in the system...



# ECDSA Challenge

https://hackmd.io/kCZFAk4nRPuTzUibyfBrDw