Nonce@Once: A Single-Trace EM Side Channel Attack on Several Constant-Time Elliptic Curve Implementations in Mobile Platforms

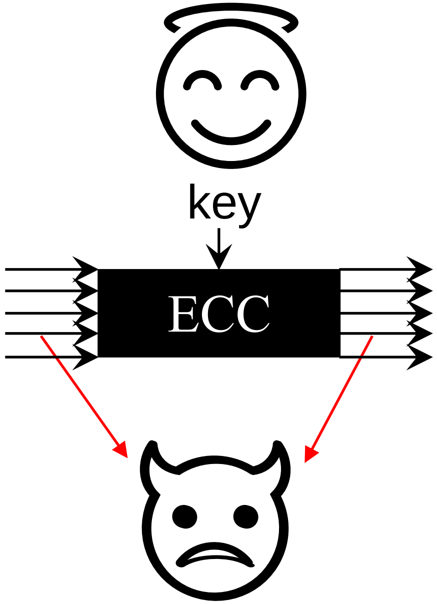
Monjur Alam, Baki Yilmaz, Frank Werner, Niels Samwel,
Alenka Zajic, Daniel Genkin, Yuval Yarom, Milos Prvulovic
Georgia Tech, Radboud University, University of Michigan, University of Adelaide
Contact: milos@cc.gatech.edu

Motivation

- Public key crypto is essential for modern security
  - Secure exchange of session keys
  - Verifying identity of systems and users
  - And much, much more
- Private keys are a highly valuable asset
  - Attackers want to get them
  - But we don’t want them to

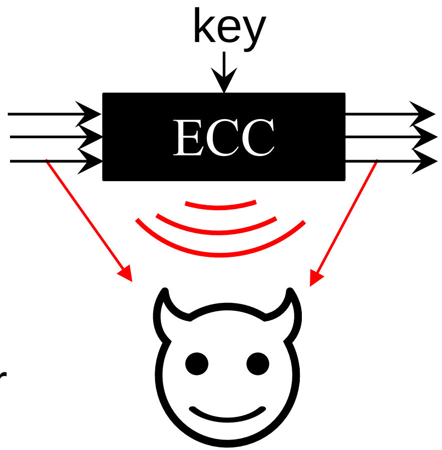
❖Public Key Crypto

- Good public key crypto (e.g. ECC)
  - Designed to make private keys very, very hard to recover



❖Analog Side-Channel Attacks

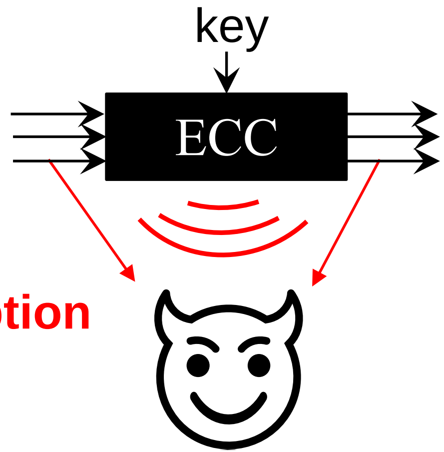
- But cryptographic implementation runs on real hardware
  - Logic gates switch, causing current flow
  - Currents flowing create changes in surrounding EM field



Most attacks:
Side-channel information helps **eventually** recover the private key

Analog Side-Channel Attacks

- But cryptographic implementation runs on real hardware
  - Logic gates switch, causing current flow
  - Currents flowing create changes in surrounding EM field



Nonce@Once:
Side-channel information
from only one signing/encryption
operation allows recovery
the private key

❖ECC Digital Signature Algorithm

1. $Q = d \cdot G$, where d is the secret key
2. $z = HASH(msg)$
3. Generate random ephemeral secret k (the “nonce”)
4. $R = k \cdot Q$
5. $r = R \rightarrow x \bmod n$
6. $s = k^{-1}(z + r \cdot d)$
7. Signature=(r, s)

Nonce k must reman secret!

If attacker knows k, a message, and its signature:
$d = (s \cdot k - z)/r \bmod n$

❖Point-by-Scalar Multiplication (R=k·Q)

```
R=Point(0);
// For each bit of nonce k
for(b=nbits-1;b>=0;b--){
        R=2·R;
        if(get_bit(k,b))
                R=R+Q;
}
```

Easy target for side channel attacks, e.g. Flush+Reload

```
R=Point(0);
// For each bit of nonce k
for(b=nbits-1;b>=0;b--){
        R=2·R;
        T=R+Q;
        Swap_Cond(R,T,get_bit(k,b));
}
```

Constant Time Implementation

❖Conditional Swap (RFC 7748)

```
Swap_Cond(A,B,cond){
        mask=0-cond;
        for(i=0;i<nwords;i++){
                Δ = (a[i]^b[i]) & mask;
                a[i]=a[i] ^ Δ;
                b[i]=b[i] ^ Δ;
        }
}
```

11..11 if cond, 00..00 if not cond

For each machine word of an EC point

$$\Delta = \begin{cases} 0, & cond == 0 \\ a\hat{\ }b, & cond == 1 \end{cases}$$

Note this is also Constant-Time!

But… ~ 40 XOR operations with Δ in Swap_Cond

All have a zero operand when cond==0

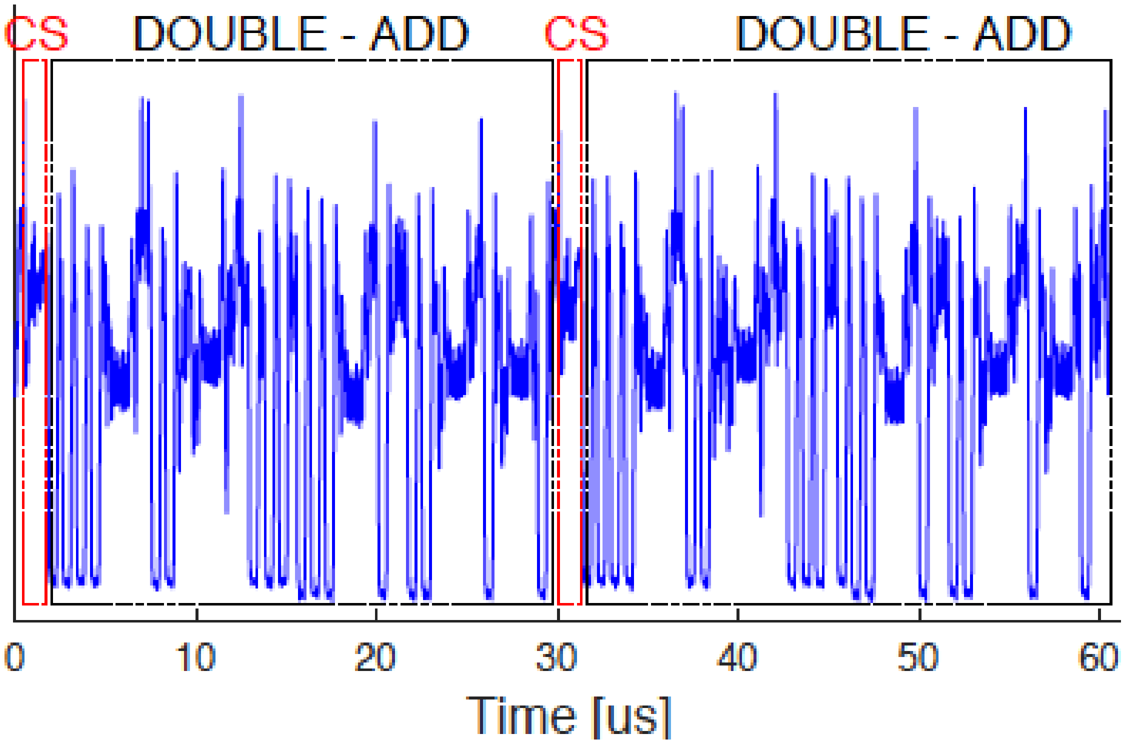
That operand is ~50%-ones when cond==1

❖Measurement Setup

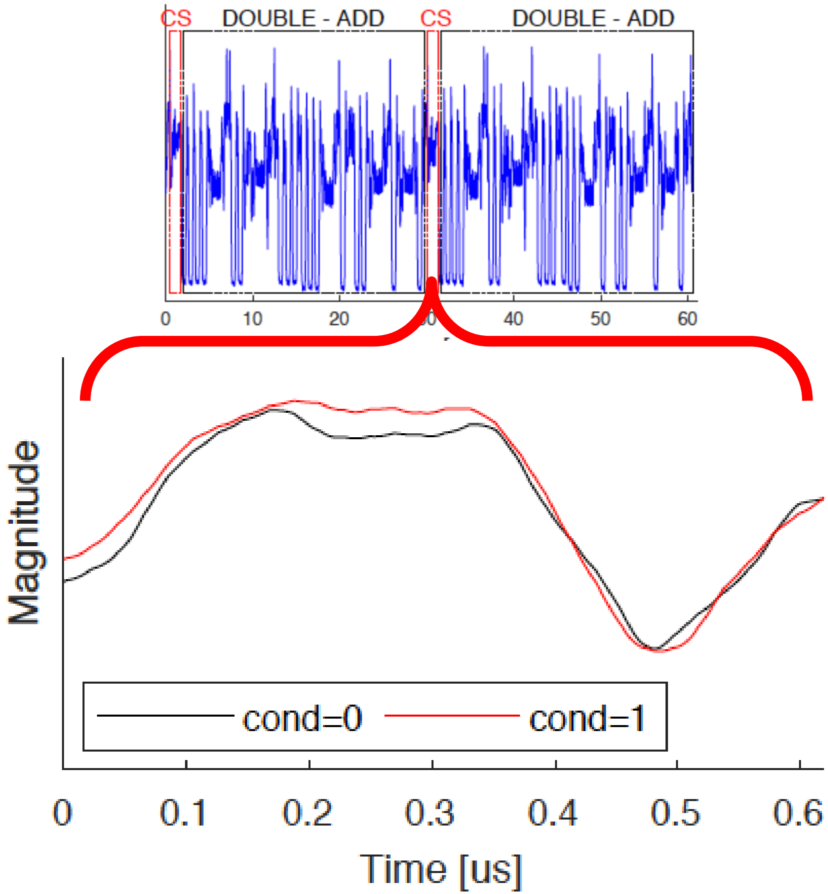
ZTE ZFIVE

Alcatel Ideal

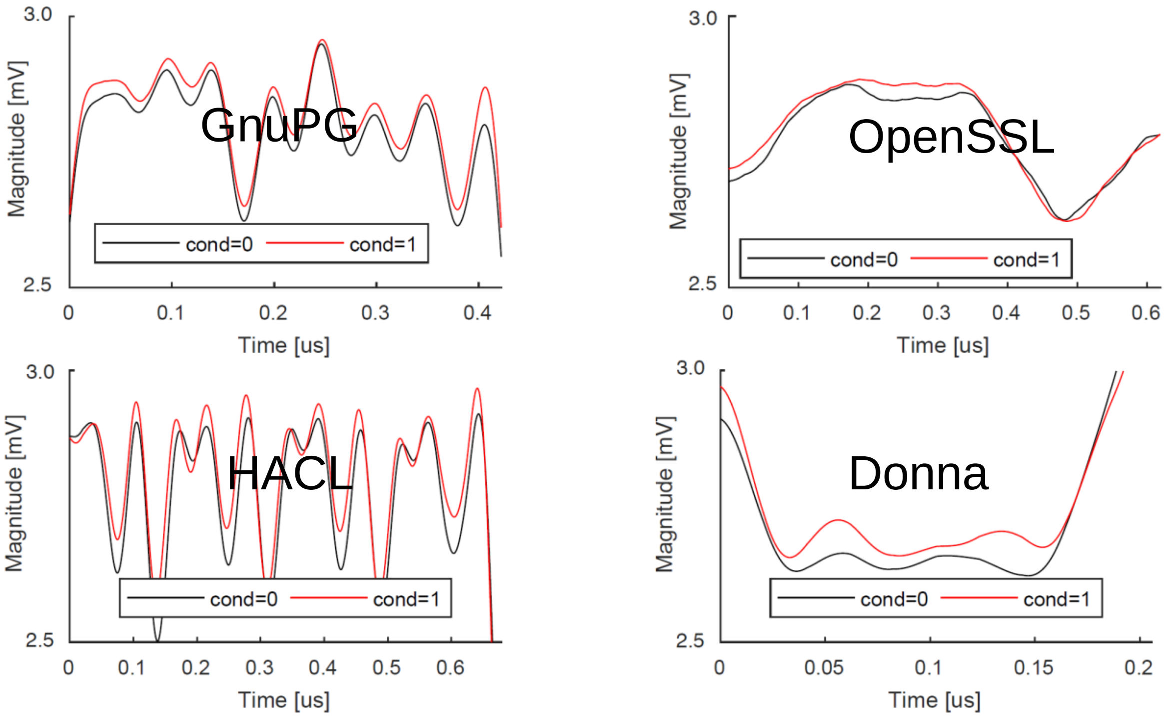
❖Locating the Cond-Swap Signals (OpenSSL)



❖Recovering value of cond (OpenSSL)



❖Recovering value of cond

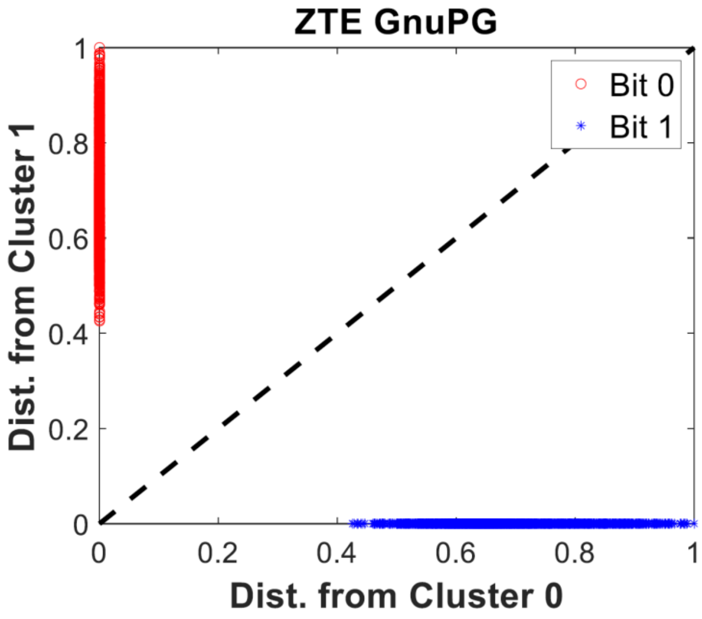


❖Nonce Recovery Algorithm

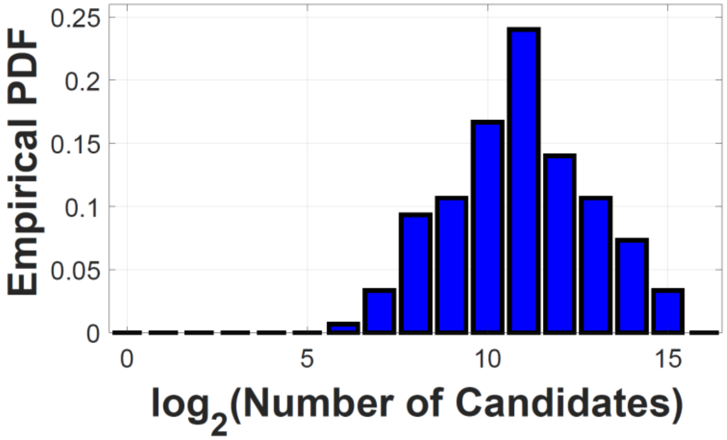
- Training
  - Record signal while signing with a few known nonces on device of same kind (but different instance of the device)
  - Cluster training Cond_Swap signals (K-Means)
  - Keep centroid and label (0 or 1) of each cluster
- Attack
  - Record signal from target device
  - Identify Cond_Swap snippets
  - Label each snippet (closest cluster)
  - Brute-force labels of “missing” snippets

❖Nonce Recovery (GnuPG on ZTE)

Bit Recovery - Clusters



Erasure Recovery – Brute Force



Mitigation

- Fundamental enabler of the attack
  - Leakage amplification
    - XOR with zero or non-zero operand leaks a little about the operand
    - But same leakage repeated 40 times in each Cond_Swap!
- Mitigation – randomization to avoid amplification

❖Mitigation

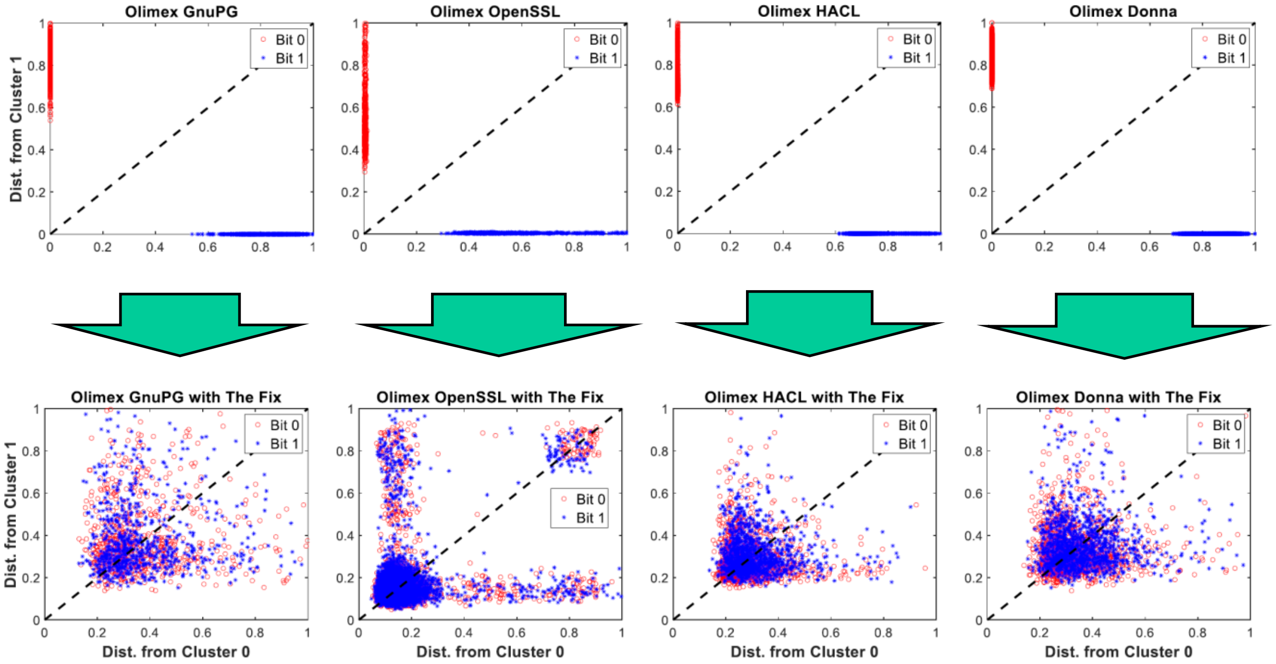
```
Swap_Cond(A,B,cond){
    mask=0-cond;
    for(i=0;i<nwords;i++){
        Δ = (a[i]^b[i]) & mask;
        a[i]=a[i] ^ Δ;
        b[i]=b[i] ^ Δ;
    }
}
```

```
Swap_Cond(A,B,cond){
    mask=0-cond;
    rand=random_word();
    for(i=0;i<nwords;i++){
        Δ′ = (a[i]^b[i]) & mask;
        Δ = Δ′ ^ rand;
        a[i]=a[i] ^ Δ ^ rand;
        b[i]=b[i] ^ Δ ^ rand;
    }
}
```

Problem: Mitigation optimized-out by compiler

Ask/trick the compiler not to do this (see paper)

❖Mitigation's Effect on the Attack



❖Conclusions

- Analog side-channel attack on constant-time ECC implementations that use conditional swap (RFC 7748)
  - Highly accurate thanks to leakage amplification
  - Successful on OpenSSL, GnuPG, HACL*, and Curve25519-donna
- ECC private key recovered from only one use of that key
- Mitigation: randomization in Cond_Swap
  - Removes leakage amplification
  - Very low performance overhead

Thank you!

Questions?