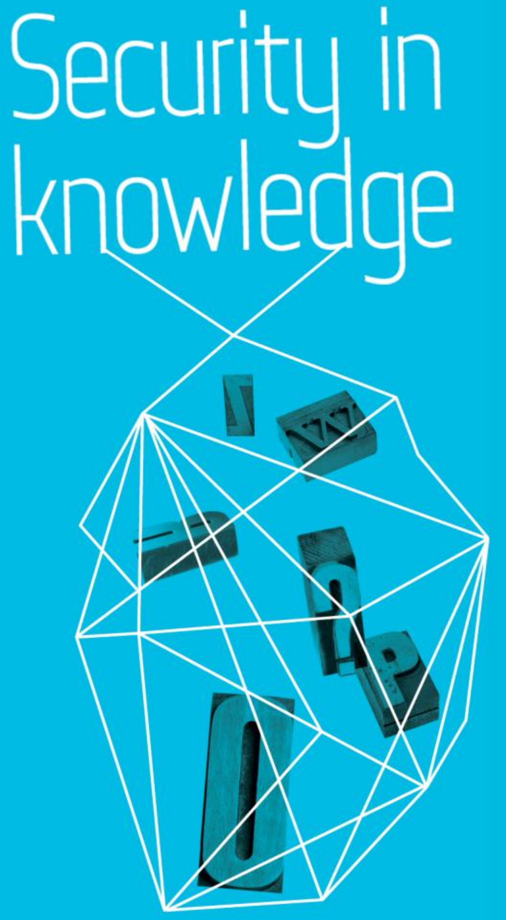
RSACONFERENCE2013



NSA Suite B Crypto, Keys, and Side Channel Attacks

Pankaj Rohatgi
Cryptography Research Inc

Mark Marson
Cryptography Research Inc

Session ID: DSP-W22

Session Classification: Advanced

What's Suite B ?

- NSA-approved cryptographic algorithms for government use (CNSSP-15)
  - http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml

- **Encryption**: AES-128 and AES-256 (FIPS PUB 197)
- **Hashing**: SHA-256 and SHA-384 (FIPS PUB 180-4)
- **Key Exchange**: ECDH using curves with 256 and 384-bit prime moduli (NIST Special Publication 800-56A)
- **Digital Signature**: ECDSA)using curves with 256 and 384-bit prime moduli (FIPS PUB 186-3)

Why Suite B matters

- Approved for protecting classified information
  - SECRET
    - AES-128, SHA-256, 256-bit ECDH, 256-bit ECDSA
  - TOP SECRET
    - AES-256, SHA-384, 384-bit ECDH,384-bit ECDSA

- Significant commercial usage:

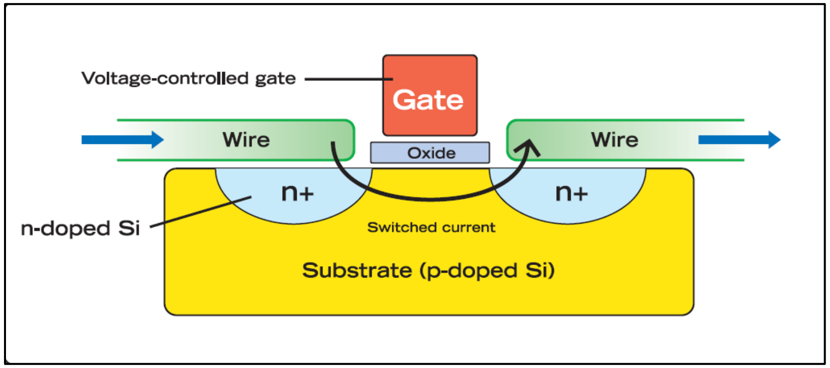
“ our …. product uses military-grade, Suite B cryptography to provide the highest level of security ”

Is it really that secure?

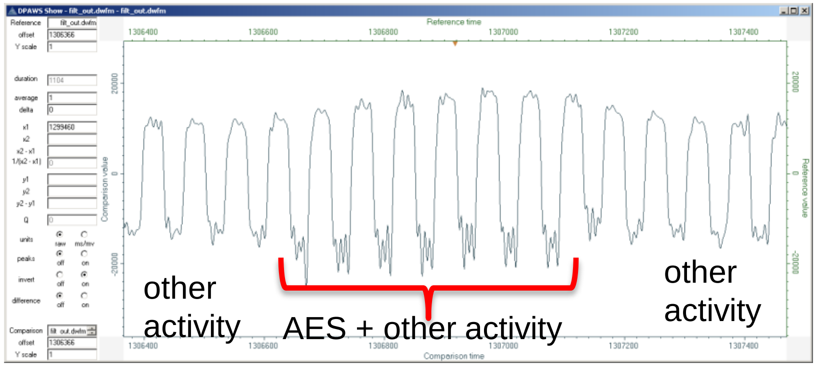
- Suite B algorithms were designed to resist all known cryptographic attacks
- But implementations have no inherent protections against non-invasive attacks !
  - E.g., side-channel attacks, fault attacks
  - Without countermeasures all Suite B implementations remain vulnerable
- We will demonstrate typical side-channel vulnerabilities in AES-128, ECDSA-384 and HMAC-SHA256.

How side channel analysis works

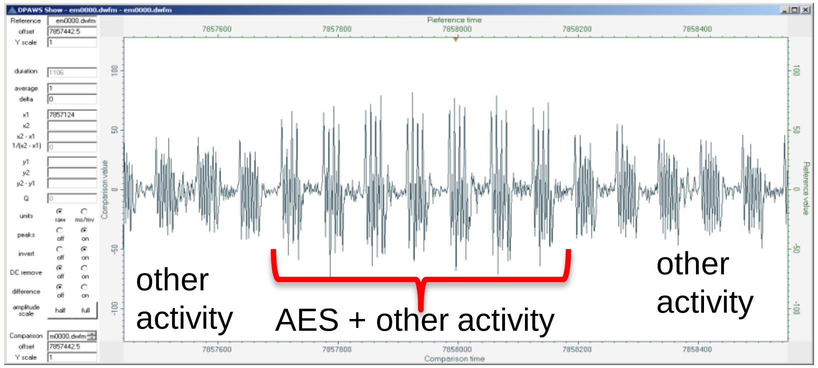
Integrated circuits contain transistors, which consume electricity as they operate. The total power consumption of an integrated circuit and its EM emissions depend on the activity of its individual transistors.



NMOS (N-Channel) Transistor



Power Trace: AES-256 decryption



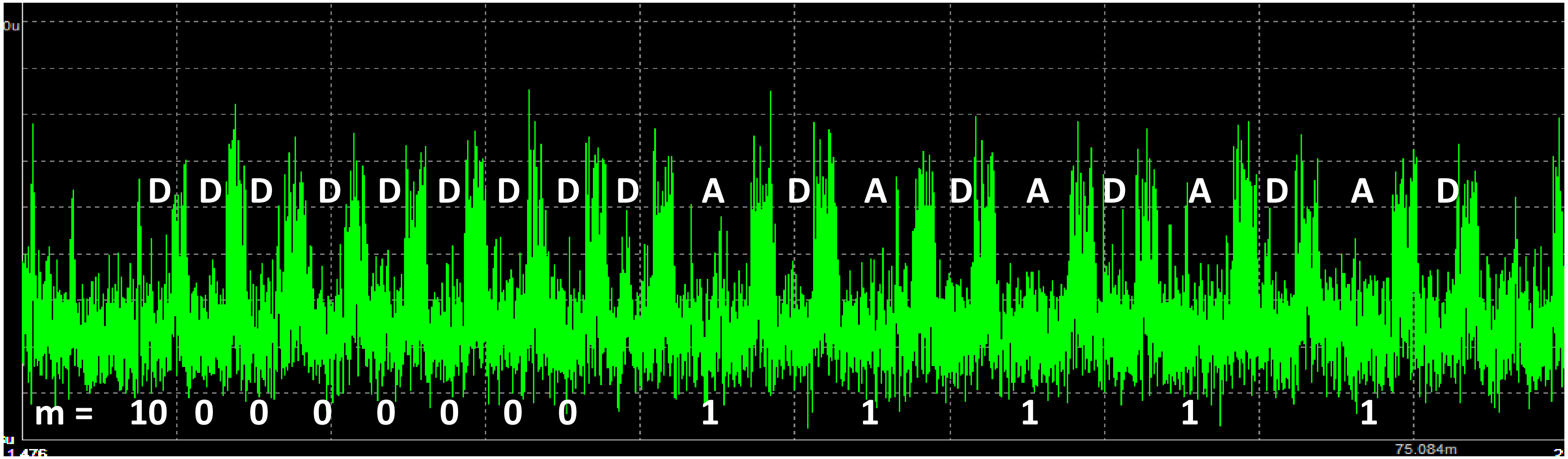
EM Trace: AES-256 decryption

Simple Power (EM) Analysis

- Keys can be recovered from a single trace

ECC: point multiplication by secret m
for each bit i of secret m
  perform **"Point Double"**
  if (bit i == 1)
    perform **"Point Add"**

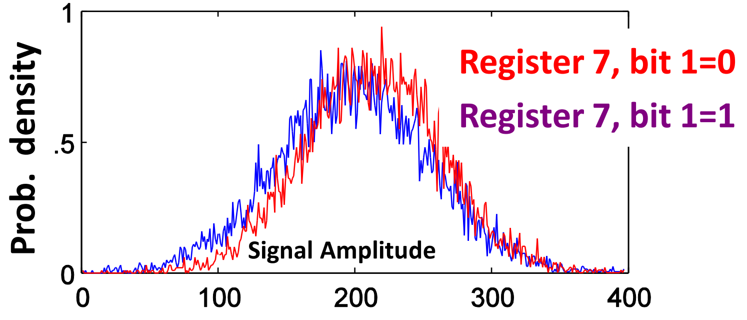
- Double-and-add algorithm to compute m*Q
- In ECC, double and add are very different operations



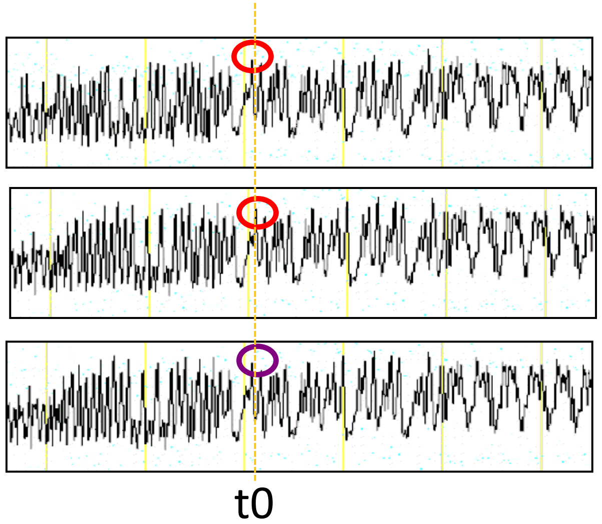
Data dependence in Power/EM traces

Data dependent differences in Power/EM traces can be quite small

- However, statistical influence remains…



Power signal amplitude at time t0



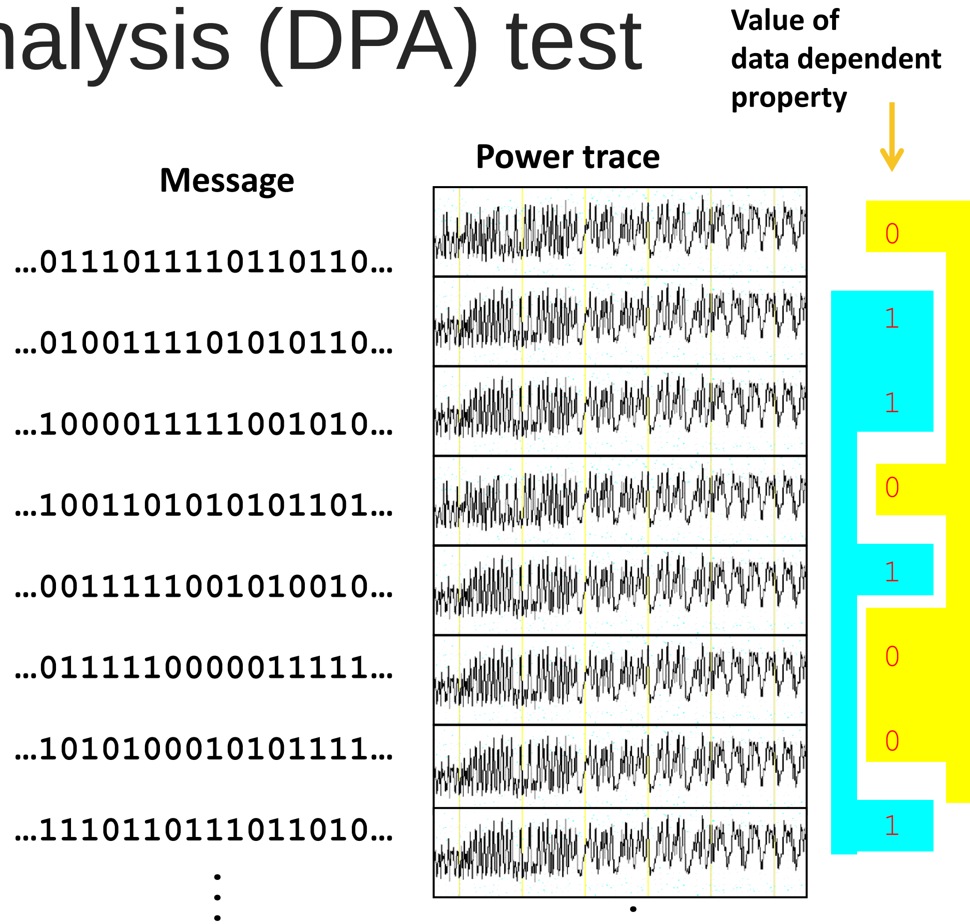
Distribution of signal amplitude where register 7 bit 1= 0 is different from distribution where register 7 bit 1 = 1

- e.g, mean is different

Differential Power Analysis (DPA) test

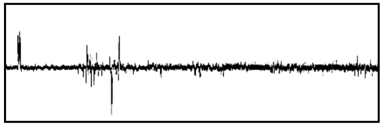
Process for testing data dependence in power/EM traces

- Perform multiple device operations with differing data
- Record power traces and corresponding data
- Partition power traces into subsets, according to **a data dependent property**
  - E.g., data bit of an intermediate state during processing
- Calculate **difference of means between the subsets**
  - Vector approach (over each time instant)



Results:

- Difference trace shows **spikes** at time offsets wherever data dependent property affects power consumption !

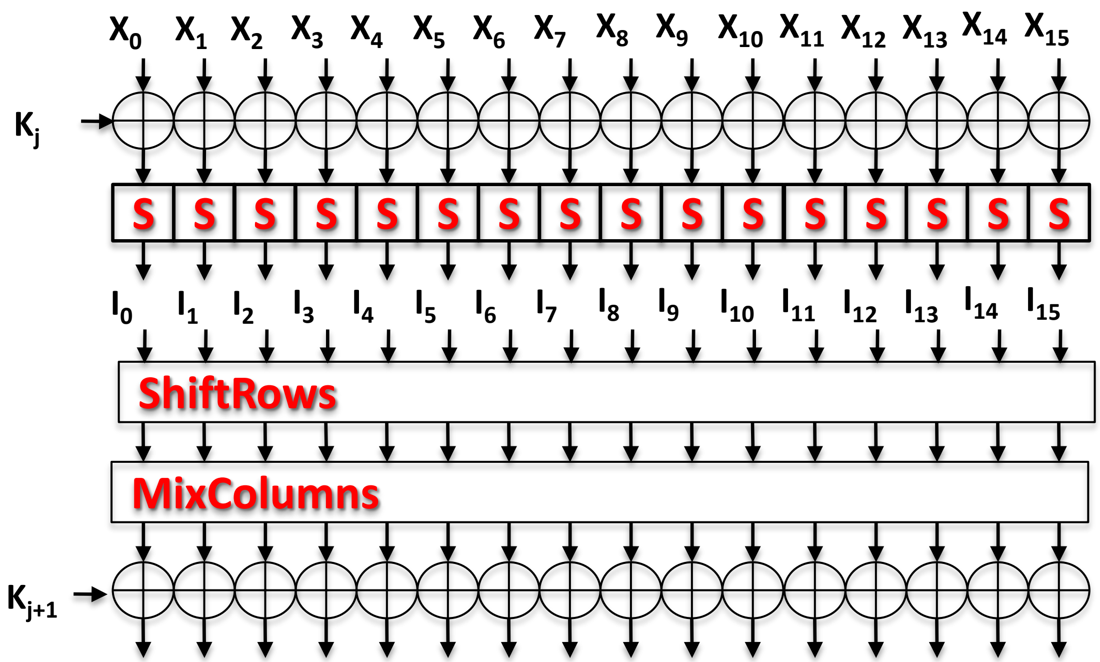


Difference trace x 25

DPA tests are extremely powerful..

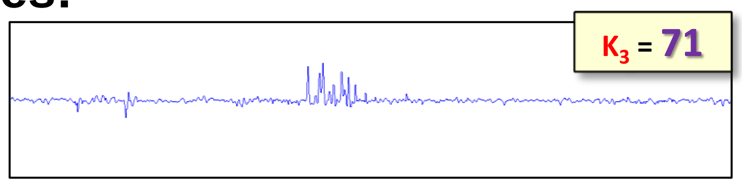
- With enough traces, a DPA test can isolate the tiniest data dependent leakage
  - Bits moving on buses, wires, switches
  - Bits written to registers
  - Bits that change when a register is overwritten
  - Switching activity in combinatorial circuits
  - A single transistor switching
  - ……..

AES

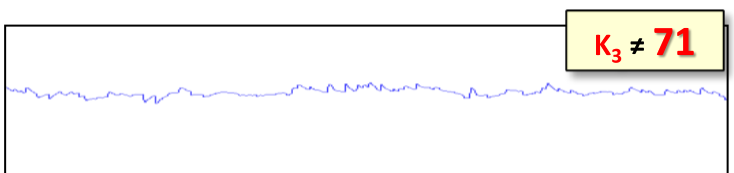


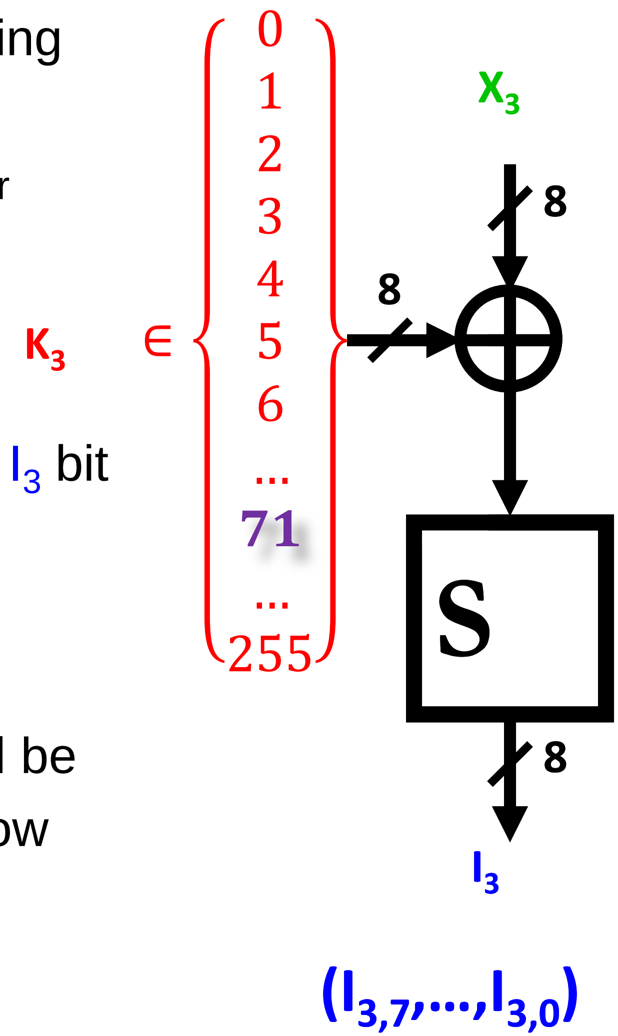
DPA attack: AES example

- DPA tests can also recover secret keys by focusing on intermediates that depend on few key bits
  - Although the key is not known, K_3 is only an 8 bit number
  - Its value must be between 0 and 255, inclusive
  - Perform 256 DPA tests on a bit of I_3 using all possible 256 guesses for the value of K_3
- For the correct guess of K_3, the DPA test on the I_3 bit will show **spikes:**



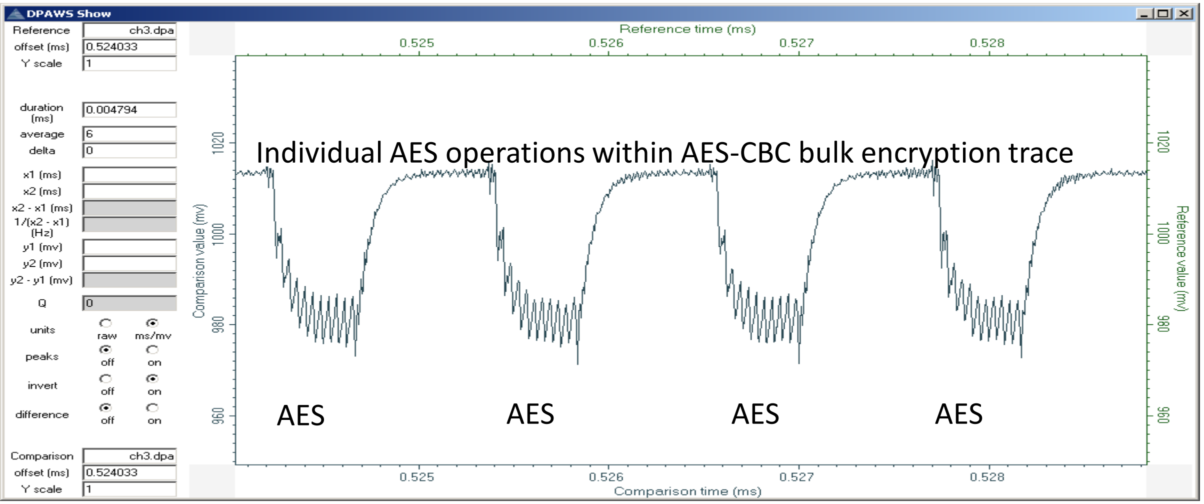
- For incorrect guesses of K_3, the derived I_3 bit will be uncorrelated to device activity: DPA tests will show **no spikes**

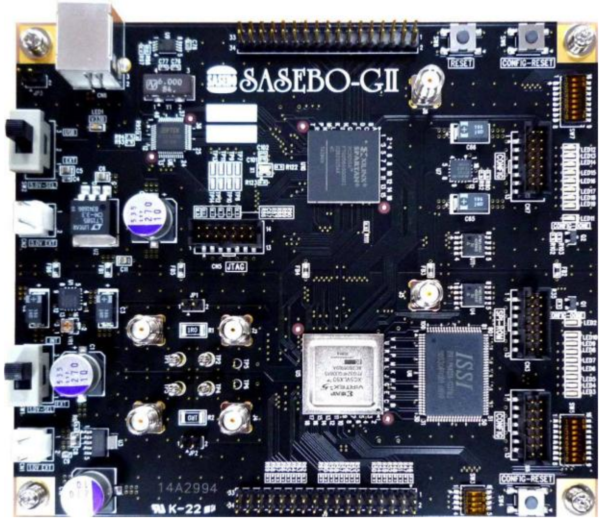




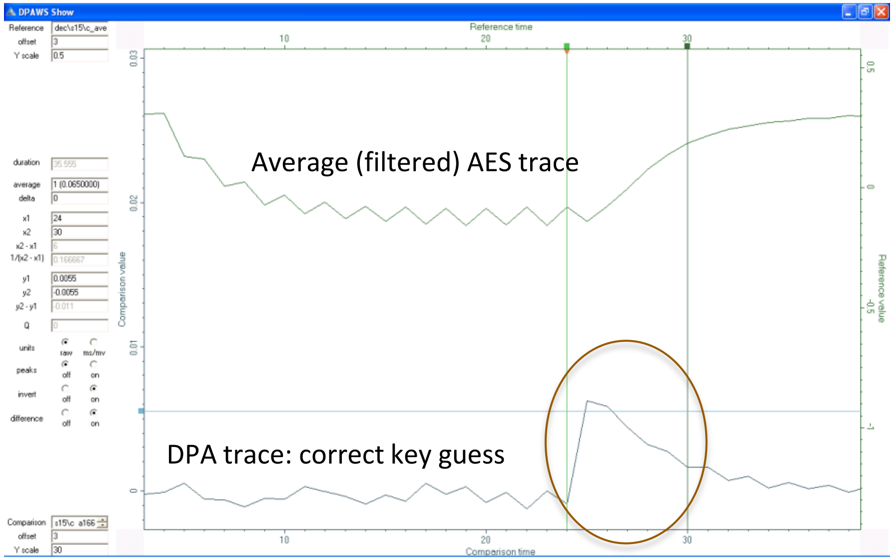
Demo: DPA on hardware AES

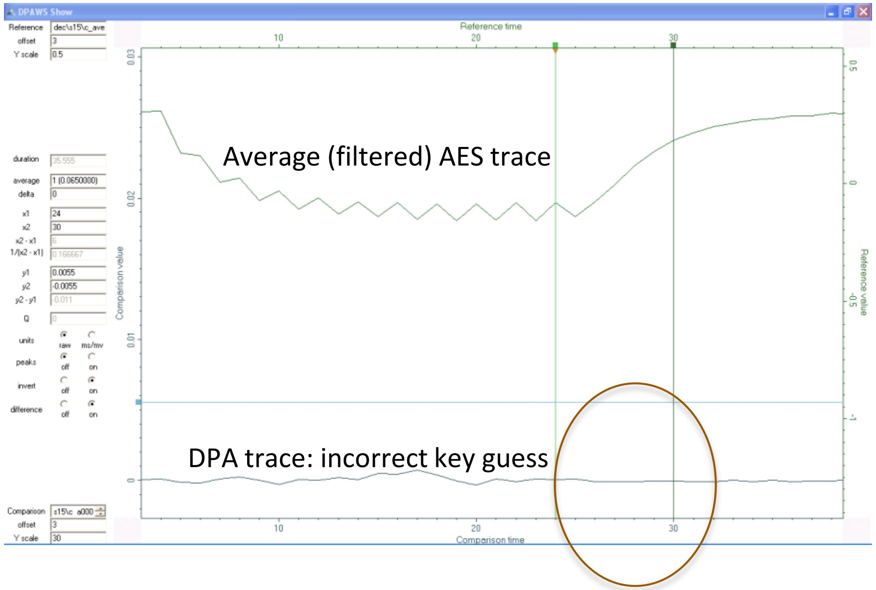
- Typically AES is implemented in hardware to perform bulk encryption
  - CBC-mode or counter-mode
  - 20k block cipher operations for a 320KB buffer
- Power trace from single bulk encryption contains multiple independent AES operations with same key
- Can use multiple AES ops from a single trace to perform DPA





Demo: Results



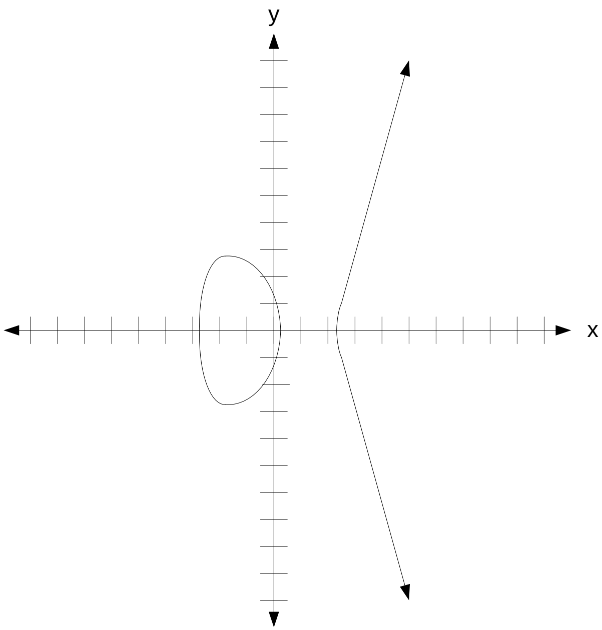


For correct key guess,
DPA test has peak

For incorrect key guess,
DPA test does not have peak

Elliptic curves over prime fields

- Elliptic curves: prime fields
  - For a and b in $\mathbf{Z}_P$, an elliptic curve is the set of all points Q = (x, y), x, y in $\mathbf{Z}_P$, which satisfy the equation:
    - $y^2 = x^3 + a*x + b \bmod P$
  - Q along with a special point O (identity, or point at infinity) forms a group with the following addition formulae



$y^2 = x^3 - 6x + 2$

- Adding distinct points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$
  - $R = P + Q = (x_3, y_3)$
  - $m = (y_2 - y_1)/(x_2 - x_1)$
  - $x_3 = m^2 - x_1 - x_2$
  - $y_3 = m(x_1 - x_3) - y_1$
- Adding a point $P = (x_1, y_1)$ to itself
  - $R = P + P = (x_3, y_3)$
  - $m = (3x_1^2 + a)/(2y_1)$
  - $x_3 = m^2 - 2x_1$
  - $y_3 = m(x_1 - x_3) - y_1$
- P + O = P = O + P,

ECDSA

- Let E be an elliptic curve over $\mathbf{F}_p$ and G an element of order n ($n * G = \mathbf{O}$) in E. Let H(M) denote the hash of the message to be signed.
- Key Generation
  - Generate a random private key a, $1 < a < n$, and compute public key $Q = a * G$.
- Signature Generation
  - Generate a random secret nonce k, $1 < k < n$, and computes $T = k * G = (x,y)$
  - Compute $r = x \bmod n$
  - Compute $s = k^{-1} * (H(M) + a * r) \bmod n$
  - Signature of M is (r, s)
- Signature Verification
  - Compute $u_1 = s^{-1} * H(M) \bmod n$ and $u_2 = = s^{-1} * r \bmod n$
  - Compute $(x,y) = u_1 * G + u_2 * Q$
  - Compute $v = x \bmod n$
  - Signature is valid if and only if $v = r$

SPA/DPA vulnerabilities in ECDSA

- Point multiplications during key generation and signature generation
  - Q = **a** * G
  - T = **k** * G

SPA leakage during point multiplication may leak information about secret scalars a and k

- Inversion of the secret nonce k
  - $k^{-1} \bmod q$

SPA leakage during may leak information about some bits k

- Multiplication of secret key with first half of signature
  - **a** * **r** mod q

Classic DPA vulnerability:
Fixed secret a, used with many different (known) r’s over multiple signatures

SPA: Point multiplication

- **Double and Add** algorithm can leak scalar (see earlier)
  - Point double and add have different formulae

- For ECDSA, more efficient algorithms using a pre-computed table of points are popular
  - E.g., Signed comb algorithm for computing T=k*G

- SPA on efficient algorithms can provide several bits of k
  - The leak is fatal
  - Lattice (LLL) based methods can recover secret key "a" from a few ECDSA signatures + corresponding bits of k

Example: Point multiplication using signed comb

For computing m*G for p-384 using a 7-teeth signed comb

- If m is even, m ← m + q, where q is the order of the curve
  - m + q fits into 385 bits (Note 385 / 7 = 55 exactly)
- Store 64 pre-computed points for a signed comb with 7 teeth

  $P_i = (2^{6*55} + b_{i,5}2^{5*55} + \ldots + b_{i,1}2^{55} + b_{i,0})*G$ where $b_{i,j} \in \{-1,1\}$

Signed_comb_point_multiply (Point G, Integer m)

$x = m_{329} || m_{274} || \ldots || m_{109} || m_{54}$

Point result = P_x

for i = 53 to 0

result = point_double(result)

$x = m_{i+275} || m_{i+220} || \ldots || m_{i+55} || m_i$

if($m_{i+330} = 1$)

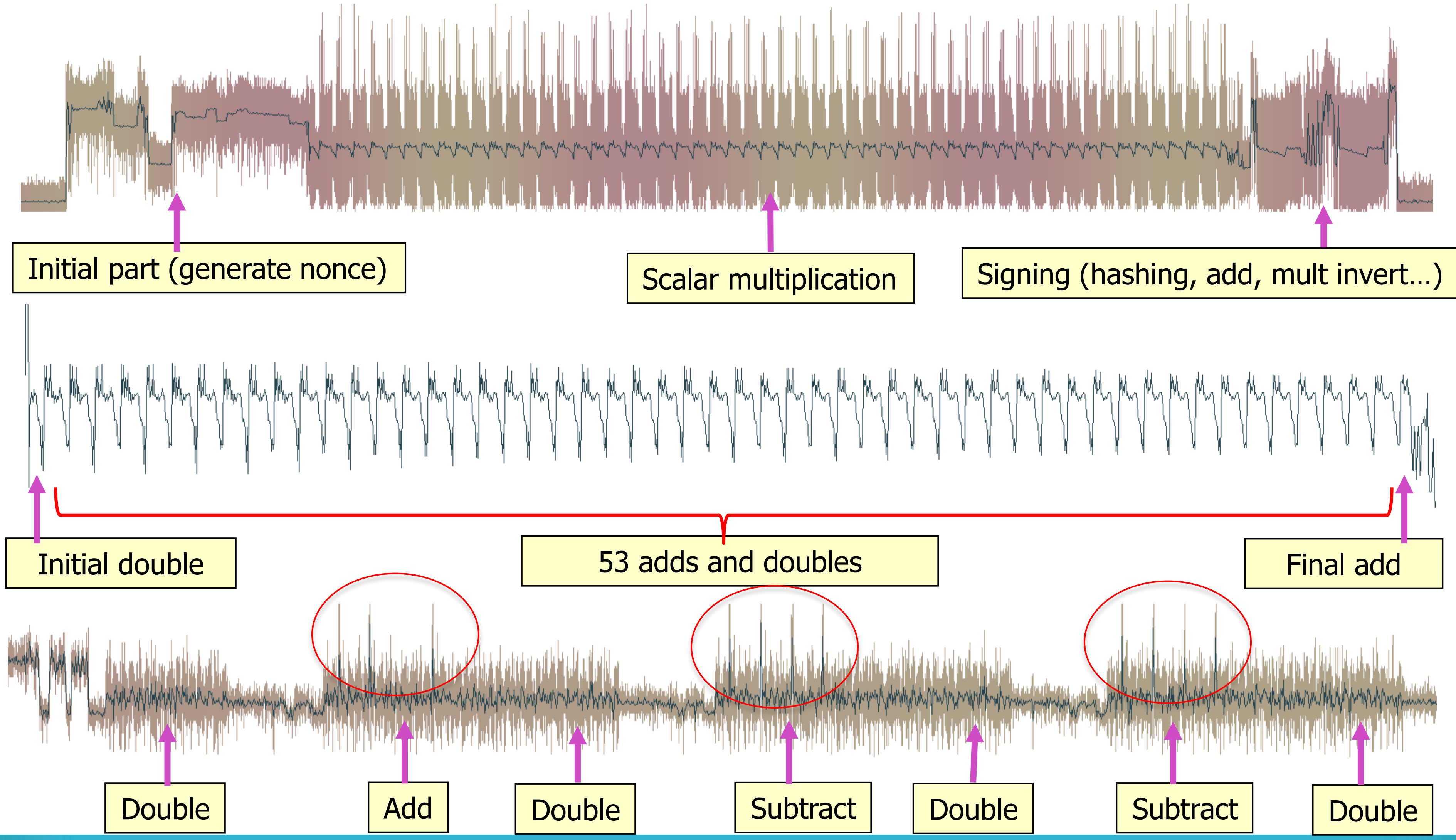
result = point_add (result, P_x)

else

result = point_subtract (result, $P_{\bar{x}}$)

Conditional operation on bits of m

Demo: ECDSA-384 on smart-card

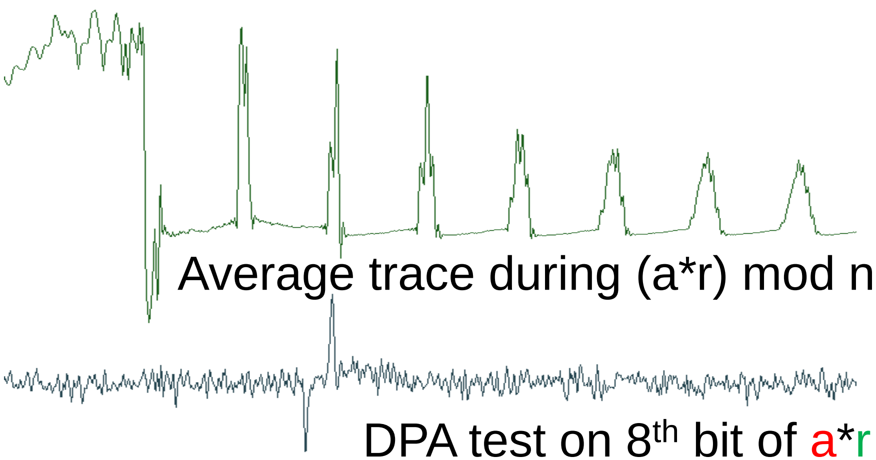


ECDSA-384: Completing the attack

- SPA reveals 54 bits of the ephemeral nonce k
  - Bits revealed: k_{i+330} $(53 \geq i \geq 0)$
- LLL attack can recover secret "a" from 9 ECDSA-384 signatures, and 54 bits of the corresponding nonces

ECDSA-384 on smart-card: Other attacks

- 6-bits of k leak during inversion of k: k → k^{-1}
  - Bleichenbacher attack to reveal secret "a"
- During $s = k^{-1} * (H(M) + a * r) \bmod n$ calculation
  - DPA tests show that all bits of (a*r) mod n leak
  - Classic DPA attack targeting secret unknown key "a", with known (random) r's.
  - 100,000 traces



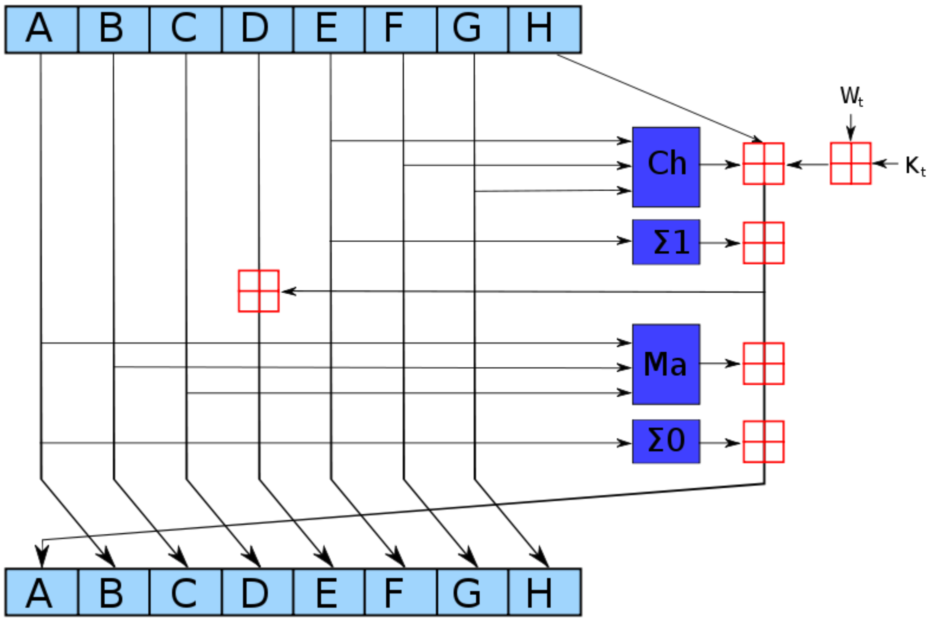
Average trace during (a*r) mod n calculation

DPA test on 8th bit of a*r mod n,
for correct guess of 12 msb's of a

SHA-256

- Arbitrary sized input
  - 512 bit input block size
  - Input blocks are "chained" together, incomplete last block is padded and length field added
  - Magic constants as initial IV, to set 8, 32-bit state variables A,B,C,D,E,F,G,H
  - Result of previous block used as "IV" for next block
- 64 rounds, 256-bit output

SHA-256 round function



Keying SHA-256

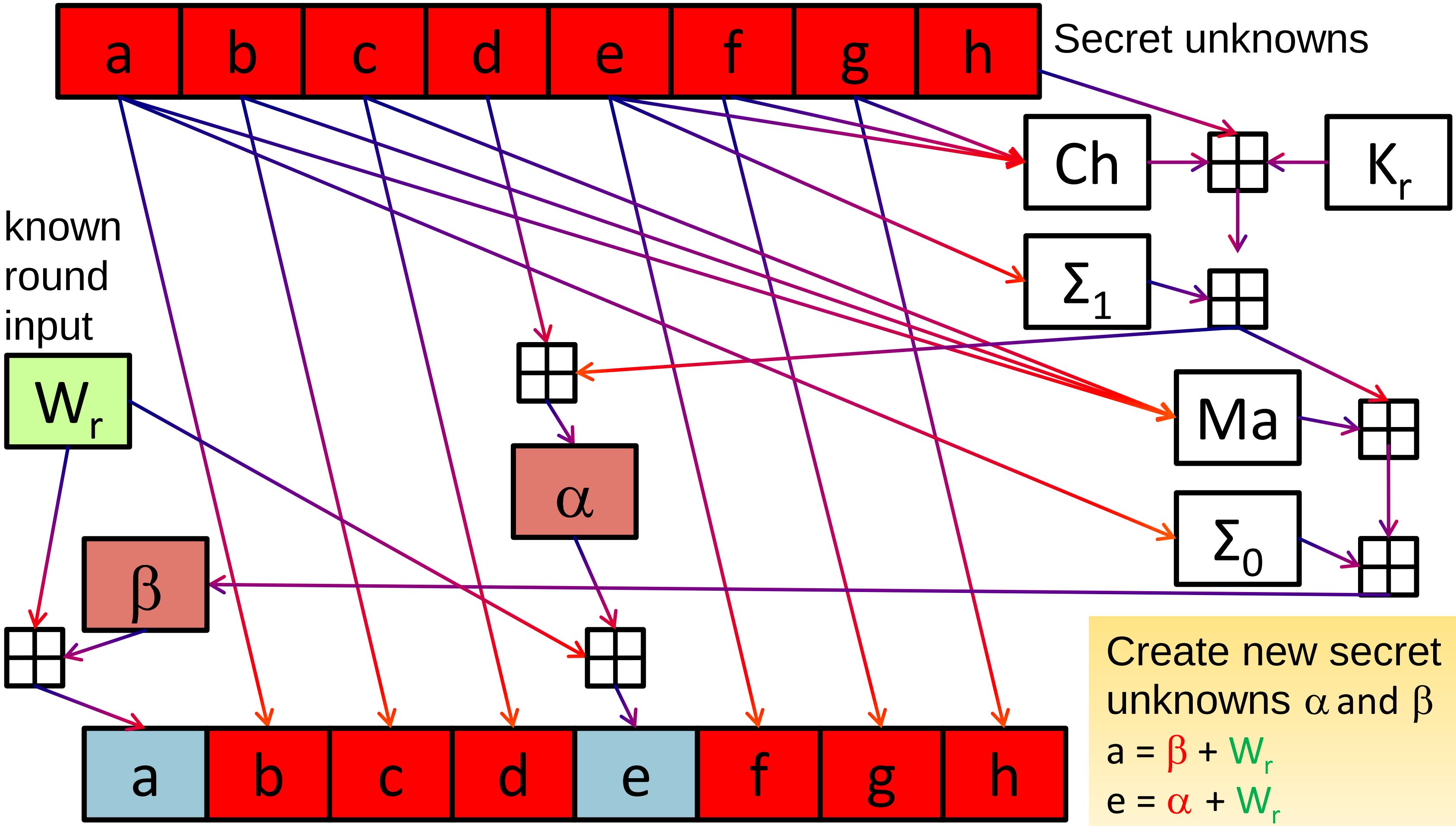
- SHA family is unkeyed

- HMAC is the standard way to key SHA to create a MAC
  - FIPS 198-1, RFC 2104, used in IPSEC, SSL, etc.

  - HMAC-SHA-256 (message)
    $= \text{SHA-256}((\text{key} \oplus \text{opad}) \,||\, \text{SHA-256}((\text{key} \oplus \text{ipad}) \,||\, \text{message}))$
    $= \text{SHA-256}_{\text{IV-Key1}}(\text{SHA-256}_{\text{IV-Key2}}(\text{message}))$

  - Essentially keys SHA-256 through its IV

DPA on inner SHA-256 in HMAC



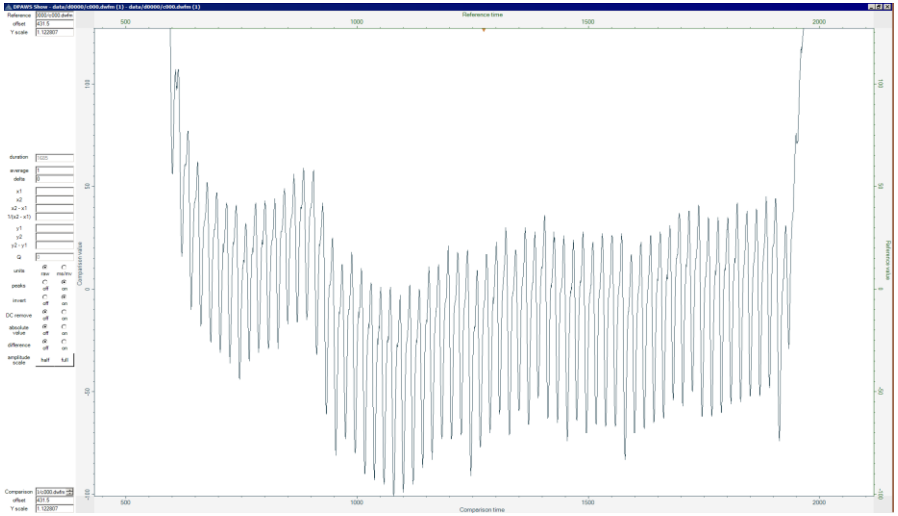
Create new secret unknowns α and β

a = β + W_r

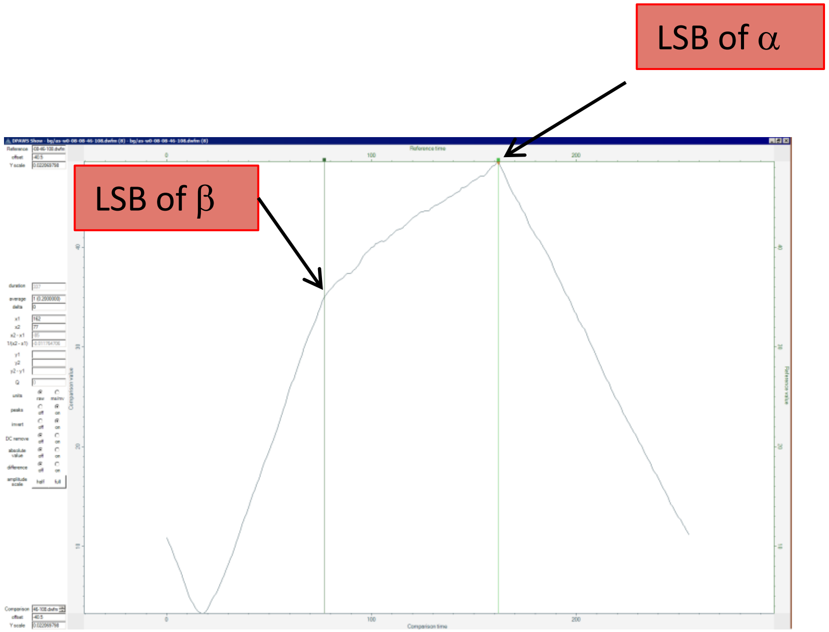
e = α + W_r

Use DPA to recover α and β for each round

SHA-256 on FPGA

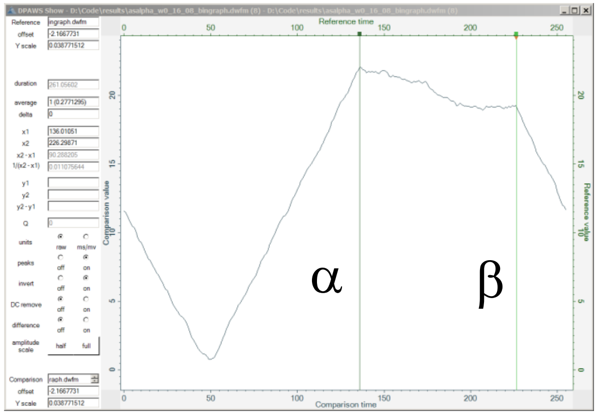


Power trace

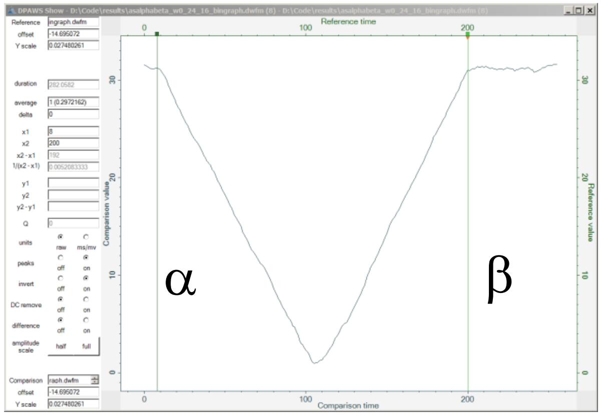


DPA attack targeting least significant byte of α and β

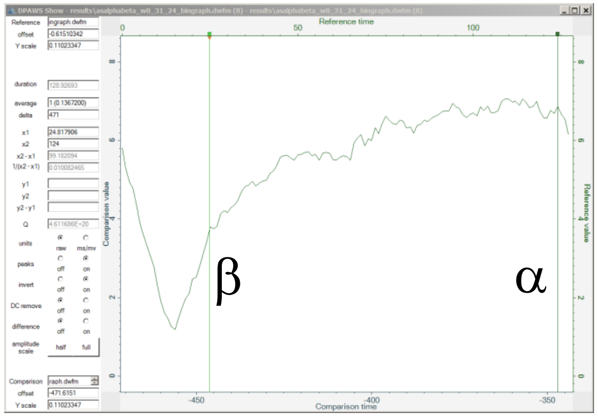
DPA to recover other bytes of α and β



Bits 16-8



Bits 24-16



Bits 31-24

- For single cycle/round implementations:
  - Use overlapping bit-ranges to distinguish bits of α from bits of β
  - Last bits of α, β and distinguishing α from β may not be possible

Completing the HMAC-SHA-256 attack

- Focus on addition and chosen messages to attack inner SHA-256
  - Fix W1, then W2 and then W3 to extract corresponding values of α, β at each round.
  - State after inputs W1, W2, W3 and W4 becomes fully known.
  - Invert to recover initial secret state a, b, c, d, e, f, g, h

- Exploit additional leakages to attack outer SHA
  - Known message scenario after the inner SHA-256 is broken

SPA/DPA countermeasures

- SPA/DPA countermeasures: fundamental categories
  - Obfuscation
  - Leak Reduction
  - Balanced HW / SW
  - Amplitude & Temporal Noise
  - Incorporating Randomness
  - Protocol Level CM
- Leak Reduction, Balanced HW/SW and Noise based countermeasures reduce attacker S/N ratio
  - No. of traces to attack proportional to $(S/N)^{-2}$
- Incorporating Randomness:
  - Computation is masked/blinding with random values to de-correlate intermediates from keys and inputs/outputs
- Protocol level countermeasures: Protocols modified to limit number of side-channels traces available per key

Example ECDSA-384

- Signed comb: **Leak reduction**
  - Store 128 pre-computed points instead of 64
  - 64 points P_x as before + 64 points $-P_{\bar{x}}$
  - Removes conditional branch in point multiplication
- **Blinding** to protect k → k^{-1}inversion and a*r mod n
  - To compute s = k^{-1}*(H(M) + a*r)
    - Generate **random blinding value** t
    - Compute k*t mod q and y = $(k*t)^{-1}$ mod q
    - Compute z = a*y = a*$(k*t)^{-1}$
    - Compute b = z*r = a*$(k*t)^{-1}$*r
    - Compute c = H(M)*y = H(M)*$(k*t)^{-1}$
    - Compute d = b + c (mod n) = $(k*t)^{-1}$*(H(M) + a*r) (mod n)
    - **Unblind** by computing t*d (mod n) = k^{-1}*(H(M) + a*r) (mod n) = s

Conclusion

- Suite B provides strong, standards based cryptography

- But, Suite B implementations have no inherent defenses against non-invasive, side-channel attacks

- Side-channel countermeasures should be adopted in all implementations that need tamper-resistance.

Questions ?

Pankaj Rohatgi

Director of Engineering

Cryptography Research Inc

415.397.0123 x4338

rohatgi@cryptography.com

Mark Marson

Technical Director

Cryptography Research Inc

415.397.0123 x4326

mark@cryptography.com