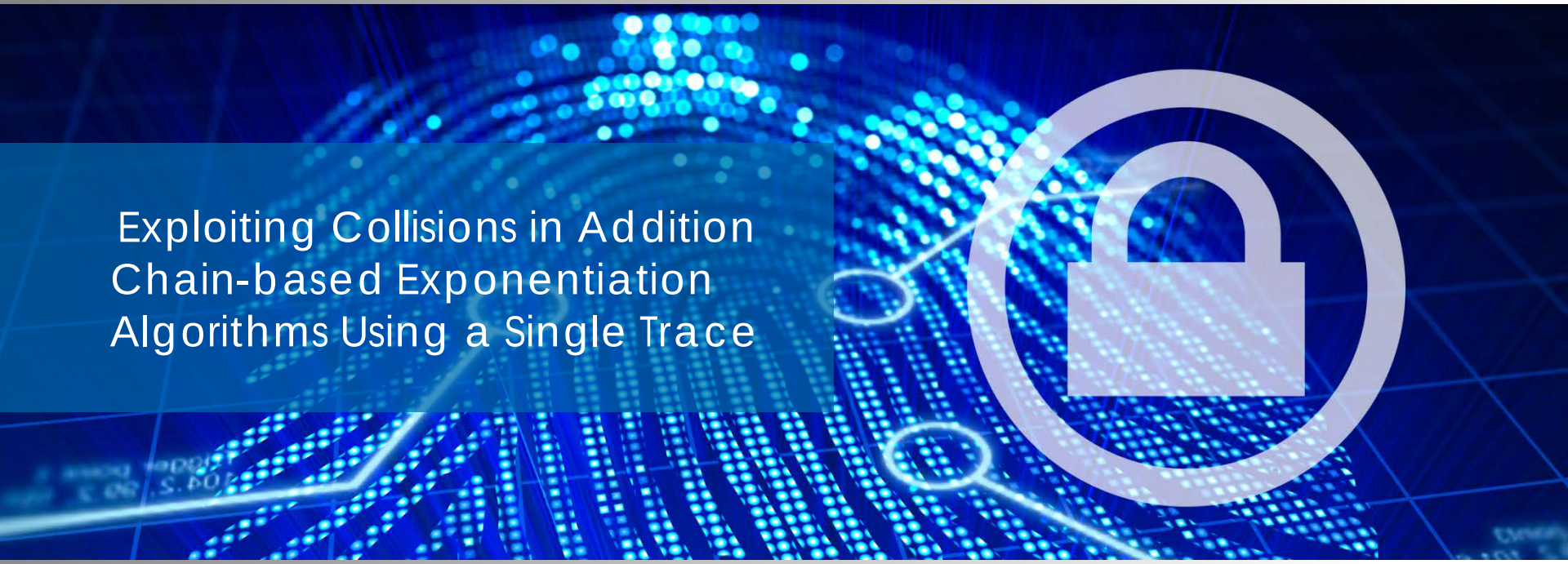
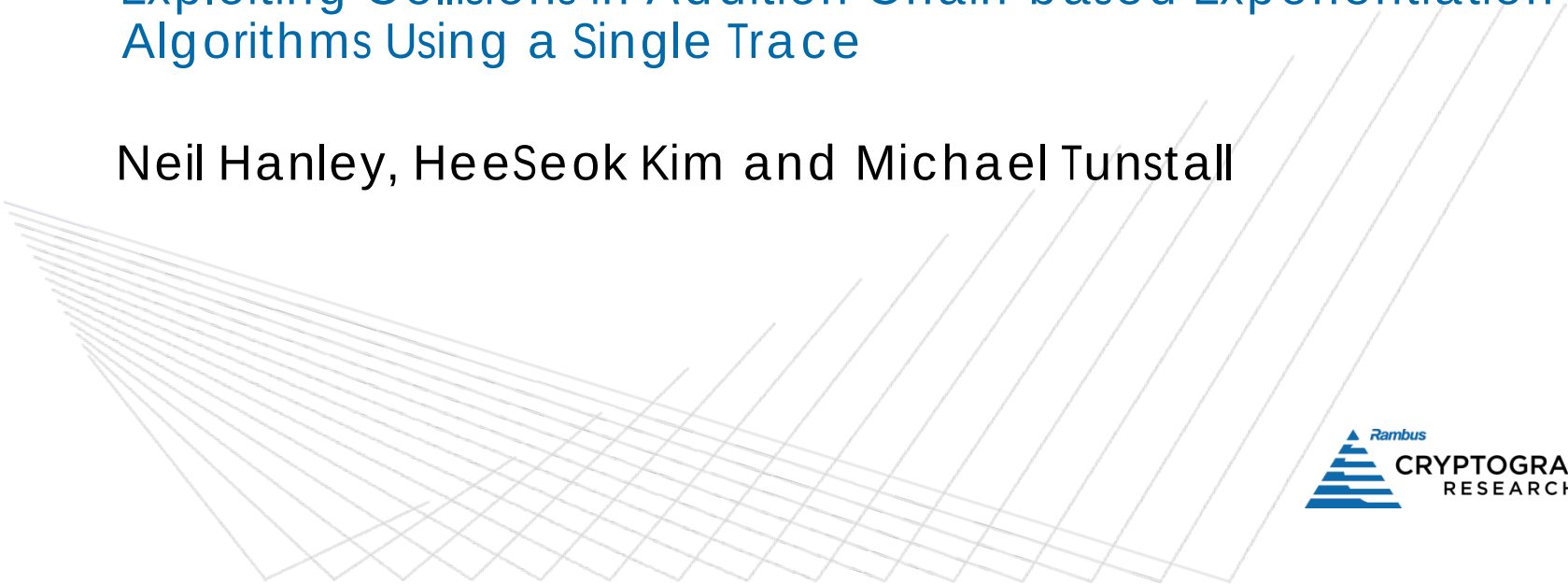
Exploiting Collisions in Addition Chain-based Exponentiation Algorithms Using a Single Trace



Rambus
CRYPTOGRAPHY RESEARCH

Exploiting Collisions in Addition Chain-based Exponentiation Algorithms Using a Single Trace

Neil Hanley, HeeSeok Kim and Michael Tunstall



Rambus
CRYPTOGRAPHY
RESEARCH

Collision Attacks

- Referred to in the literature as:
  - Bigmac attack (Walter 2002)
  - Horizontal-Vertical attacks
  - Correlation-collision attack
- Class of attacks looking for intermediate values that are the same at two points in an algorithm
  - Identical operand(s) for operations
  - Result of one operation being used as the input to another operation
- We describe versions of these attacks applied to a single trace
  - Applicable to blinded/ephemeral exponents

Algorithm 1: Joye's Add-Only Scalar Multiplication

Input: $\boldsymbol{P}$ a point on elliptic curve $\mathcal{E}$, an n-bit scalar $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$

Output: $\boldsymbol{Q} = k\,\boldsymbol{P}$

1 $\boldsymbol{R}_0 \leftarrow \boldsymbol{\mathcal{O}}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_2 \leftarrow \boldsymbol{P}$;
2 **for** $i \leftarrow 0$ **to** $n-1$ **do**
3 $\quad \boldsymbol{R}_{1-k_i} \leftarrow \boldsymbol{R}_{1-k_i} + \boldsymbol{R}_2$;
4 $\quad \boldsymbol{R}_2 \leftarrow \boldsymbol{R}_0 + \boldsymbol{R}_1$;
5 **end**
6 **return** $\boldsymbol{R}_0$

- We note that R_0 in round i ...

Algorithm 1: Joye's Add-Only Scalar Multiplication

Input: $\boldsymbol{P}$ a point on elliptic curve $\mathcal{E}$, an n-bit scalar $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$

Output: $\boldsymbol{Q} = k\,\boldsymbol{P}$

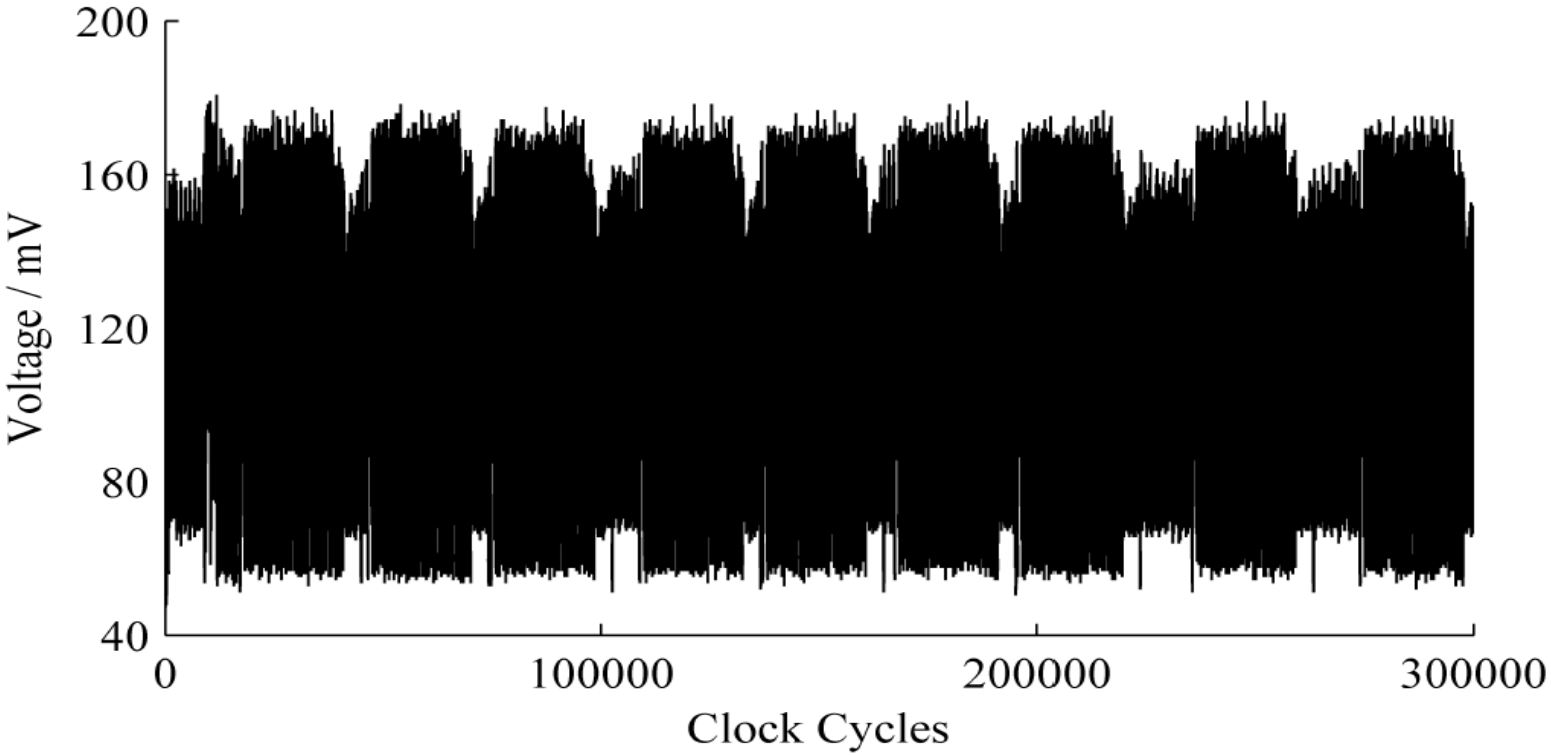
1 $\boldsymbol{R}_0 \leftarrow \boldsymbol{\mathcal{O}}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_2 \leftarrow \boldsymbol{P}$;
2 **for** $i \leftarrow 0$ **to** $n-1$ **do**
3 $\quad \boldsymbol{R}_{1-k_i} \leftarrow \boldsymbol{R}_{1-k_i} + \boldsymbol{R}_2$;
4 $\quad \boldsymbol{R}_2 \leftarrow \boldsymbol{R}_0 + \boldsymbol{R}_1$;
5 **end**
6 **return** $\boldsymbol{R}_0$

- We note that R_0 in round i, will be the same as the first operand of the first operation in round $i+1$.

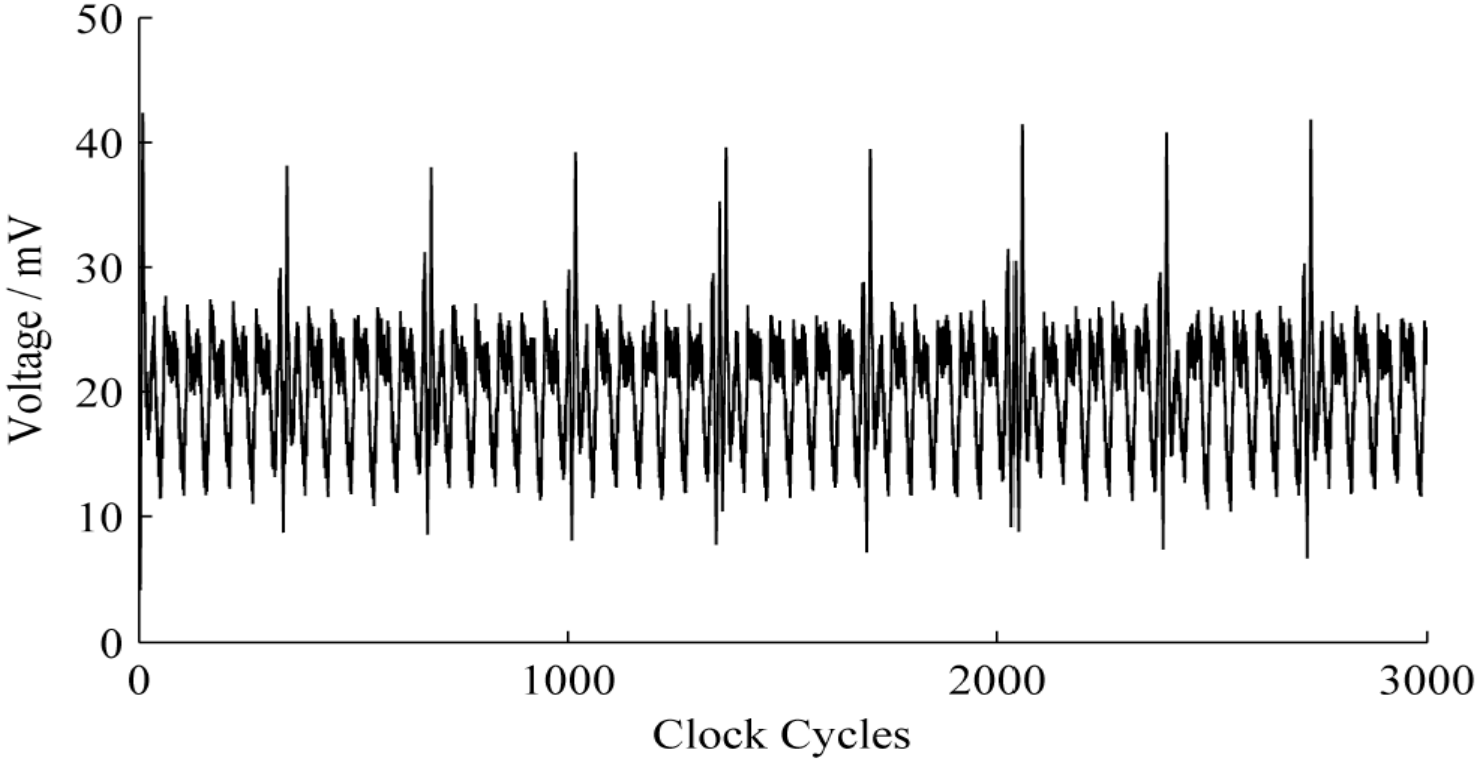
Implementing an Attack

- We considered a single trace, taken during a 192-bit scalar multiplication
  - Unknown (blinded) message
  - Single ephemeral exponent
  - Corresponding to an implementation of ECDSA
- First platform ARM7TDMI
  - Clocked at 7.37 MHz
  - Assembly implementation of basic operations
- Second platform SASEBO-G
  - Clocked at 24 MHz
  - Implementation in VHDL

Sample Trace – ARM7TDMI



Sample Trace – SASEBO-G



Attacking Joye's Add-Only Algorithm

- Break a trace into subtraces corresponding to individual operations

$$\{O_1, O_2, O_3, \dots, O_m\}$$

- Generate a mean subtrace and subtract it from each subtrace.

$$\{O'_1, O'_2, O'_3, \dots, O'_m\}$$

- We denote each these w-point subtraces as

$$A = \{a_{1,1} \dots a_{1,u}, a_{2,1} \dots a_{2,u}, \dots, a_{m,1} \dots a_{m,u}\}$$

where: $O'_i = a_{i,1} \dots a_{i,u} - \hat{a}_1 \dots \hat{a}_u = \bar{a}_i$

- Generate a trace of correlation coefficients of the same size as the subtraces, assuming that the collision occurs in every round.

$$C = \rho\big((\bar{a}_2, \bar{a}_4, \dots, \bar{a}_{m-2}), (\bar{a}_3, \bar{a}_5, \dots, \bar{a}_{m-1})\big)$$

Attacking Joye's Add-Only Algorithm

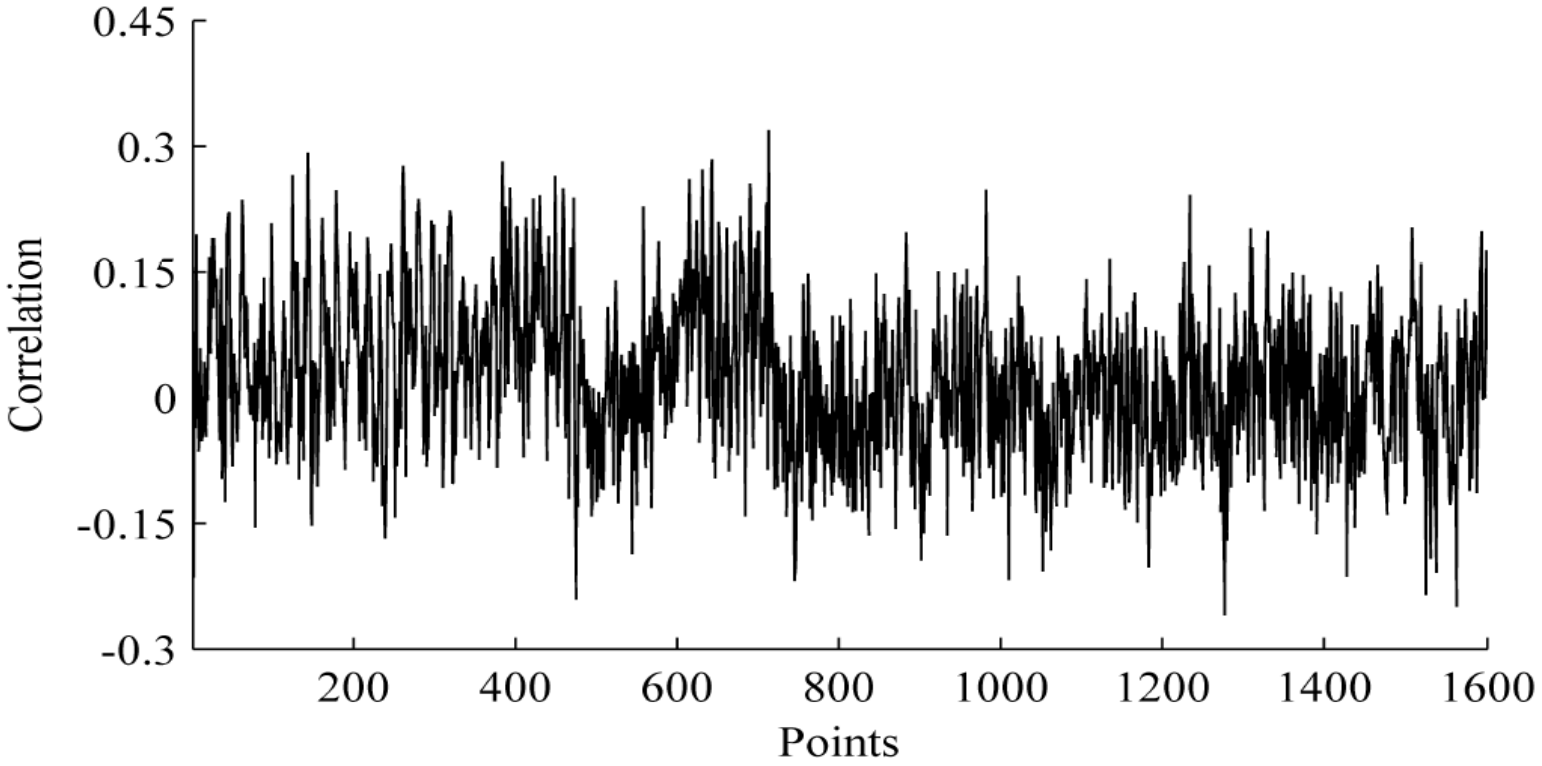
- There will be correlation indicating points that bear information
- Many points will have spurious correlation
  - Counters etc.

- To prevent this we randomly sort the set of subtraces and generate a second correlation trace

$$C' = \rho\left((\bar{a}_2, \bar{a}_4, \ldots, \bar{a}_{m-2}), (\bar{a}_3, \bar{a}_5, \ldots, \bar{a}_{m-1})\right)$$

- Subtract one correlation trace from the other point-by-point

Example Correlation Trace



Attacking Joye's Add-Only Algorithm

- We note the index of points that appear to correlate when the two operands are the same (at least some of the time)
- Extract the relevant points from each subtrace and use these points to determine whether there is any correlation for each pair of subtraces that could indicate a collision.

$$D = d_1 \dots d_{n-1} = \{\rho(\bar{a}'_2, \bar{a}'_3), \rho(\bar{a}'_4, \bar{a}'_5), \dots, \rho(\bar{a}'_{m-2}, \bar{a}'_{m-1}\}$$

- Generates a correlation coefficient for each bit that we split into hypotheses by comparing them to the mean correlation.

Algorithm 2: Coron's Double-and-Add-Always Algorithm

Input: $\boldsymbol{P}$ a point on elliptic curve $\mathcal{E}$, an n-bit scalar $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$

Output: $\boldsymbol{Q} = k\,\boldsymbol{P}$

1 $\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
2 **for** $i \leftarrow n-2$ **down to** 0 **do**
3 $\quad \boldsymbol{R}_0 \leftarrow 2\,\boldsymbol{R}_0$;
4 $\quad \boldsymbol{R}_{1-k_i} \leftarrow \boldsymbol{R}_0 + \boldsymbol{P}$;
5 **end**
6 **return** $\boldsymbol{R}_0$

- Again, we note that R_0 in round i ...

Algorithm 2: Coron's Double-and-Add-Always Algorithm

Input: $\boldsymbol{P}$ a point on elliptic curve $\mathcal{E}$, an n-bit scalar $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$

Output: $\boldsymbol{Q} = k\,\boldsymbol{P}$

1 $\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
2 **for** $i \leftarrow n-2$ **down to** 0 **do**
3 $\quad \boldsymbol{R}_0 \leftarrow 2\,\boldsymbol{R}_0$;
4 $\quad \boldsymbol{R}_{1-k_i} \leftarrow \boldsymbol{R}_0 + \boldsymbol{P}$;
5 **end**
6 **return** $\boldsymbol{R}_0$

- Again, we note that R_0 in round i, will be the same as the first operand of the first operation in round $i+1$, if round i contains a dummy operation.
- However, addition and doubling are different operations

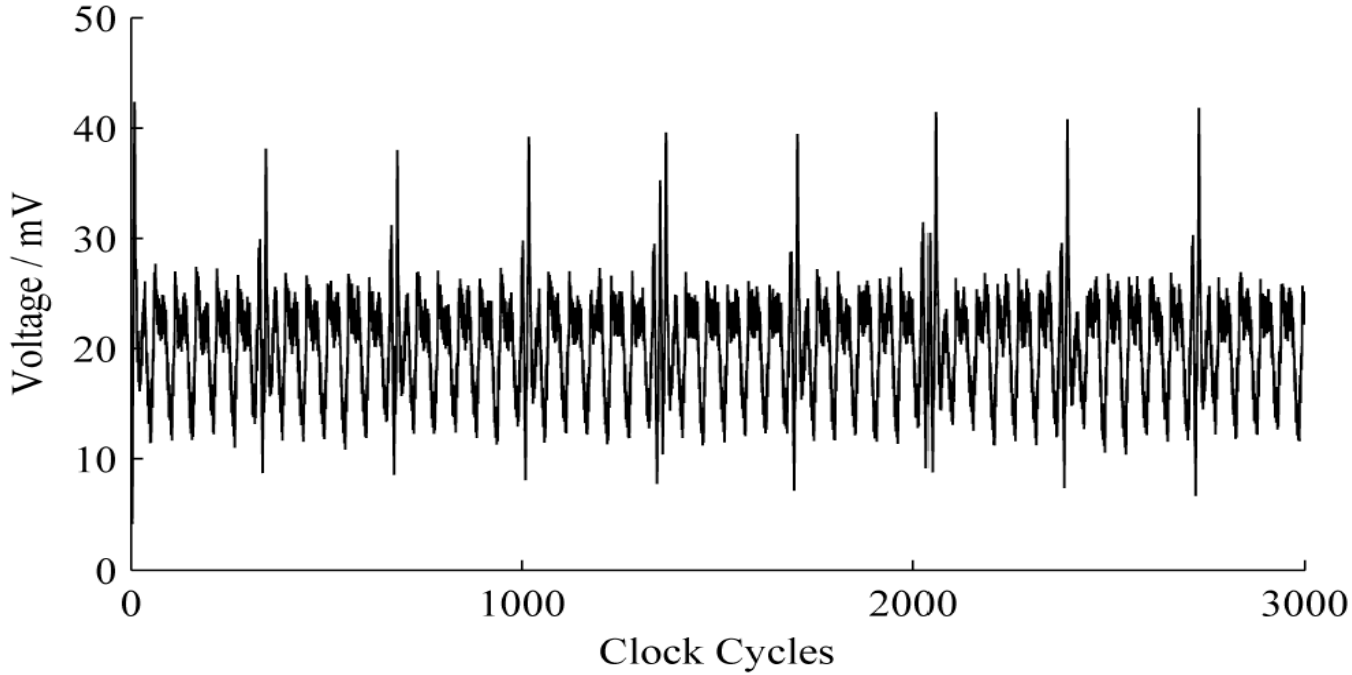
Attacking Coron's Double and Add Always

- We construct a matrix of correlation coefficients comparing all combinations of field multiplications
- We correlate traces assuming there is always a dummy operation
- As previously, we are looking at the correlation of traces taken during two consecutive operations

$$C = \rho\big((\bar{a}_2, \bar{a}_4, \dots, \bar{a}_{m-2}), (\bar{a}_3, \bar{a}_5, \dots, \bar{a}_{m-1})\big)$$

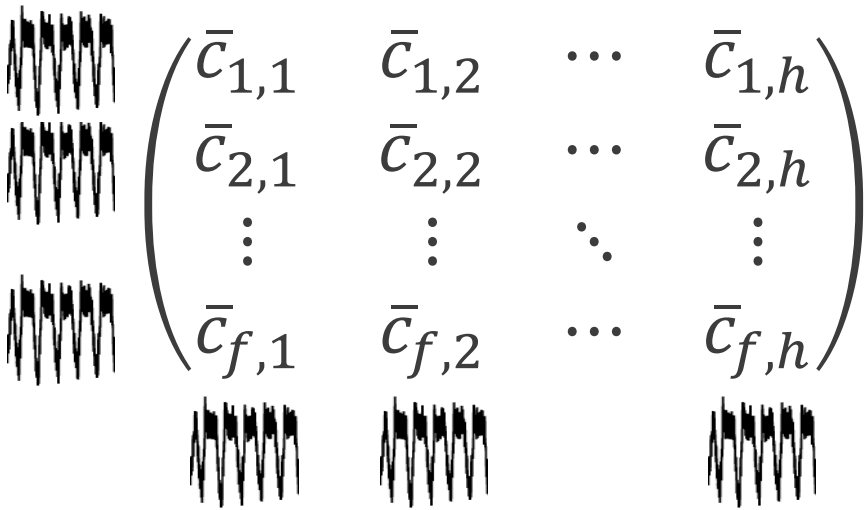
- However, an addition and doubling operation are typically different operations.
- If, for example, an addition has h, and a doubling operation has f field multiplications.

Sample Trace – SASEBO-G



- We extract each subtraces corresponding to each individual field multiplication.

Attacking Coron's Double and Add Always



- Where each $\bar{c}_{i,j}$, for $1 \leq i \leq f$ and $1 \leq j \leq h$, is a u-point trace representing a field multiplication.

Attacking Coron's Double and Add Always

- Construct a matrix assuming a dummy operation always occurs, and another where the random operations are compared
- The difference indicating what combination of points can detect a collision
- Applied to subtraces from individual loops giving correlation coefficients that can be use to determine bits of the scalar as previously

$$D = d_1 \dots d_{n-1} = \{\rho(\bar{a}'_2, \bar{a}'_3), \rho(\bar{a}'_4, \bar{a}'_5), \dots, \rho(\bar{a}'_{m-2}, \bar{a}'_{m-1}\}$$

Algorithm 3: Montgomery Ladder

Input: $\boldsymbol{P}$ a point on elliptic curve $\mathcal{E}$, an n-bit scalar $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$

Output: $\boldsymbol{Q} = k\,\boldsymbol{P}$

1 $\boldsymbol{R}_0 \leftarrow \boldsymbol{\mathcal{O}}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
2 **for** $i = n - 1$ **down to** 0 **do**
3 $\quad \boldsymbol{R}_{\neg k_i} \leftarrow \boldsymbol{R}_{k_i} + \boldsymbol{R}_{\neg k_i}$;
4 $\quad \boldsymbol{R}_{k_i} \leftarrow 2\,\boldsymbol{R}_{k_i}$;
5 **end**
6 **return** $\boldsymbol{R}_0$;

- We note that $R_{\neg k_i}$ in round i …

Algorithm 3: Montgomery Ladder

Input: $\boldsymbol{P}$ a point on elliptic curve $\mathcal{E}$, an n-bit scalar $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$

Output: $\boldsymbol{Q} = k\,\boldsymbol{P}$

1 $\boldsymbol{R}_0 \leftarrow \boldsymbol{\mathcal{O}}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
2 **for** $i = n - 1$ **down to** 0 **do**
3 $\quad \boldsymbol{R}_{\neg k_i} \leftarrow \boldsymbol{R}_{k_i} + \boldsymbol{R}_{\neg k_i}$;
4 $\quad \boldsymbol{R}_{k_i} \leftarrow 2\,\boldsymbol{R}_{k_i}$;
5 **end**
6 **return** $\boldsymbol{R}_0$;

- We note that $R_{\neg k_i}$ in round i is used as an input to the second operation in round $i + 1$, if $k_i \neq k_{i+1}$.

Algorithm 3: Montgomery Ladder

Input: $\boldsymbol{P}$ a point on elliptic curve $\mathcal{E}$, an n-bit scalar $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$

Output: $\boldsymbol{Q} = k\,\boldsymbol{P}$

1 $\boldsymbol{R}_0 \leftarrow \boldsymbol{\mathcal{O}}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
2 **for** $i = n - 1$ **down to** 0 **do**
3 $\quad \boldsymbol{R}_{\neg k_i} \leftarrow \boldsymbol{R}_{k_i} + \boldsymbol{R}_{\neg k_i}$;
4 $\quad \boldsymbol{R}_{k_i} \leftarrow 2\,\boldsymbol{R}_{k_i}$;
5 **end**
6 **return** $\boldsymbol{R}_0$;

- Likewise, R_{k_i} in round i is used as an input to the second operation in round $i + 1$, if $k_i = k_{i+1}$.

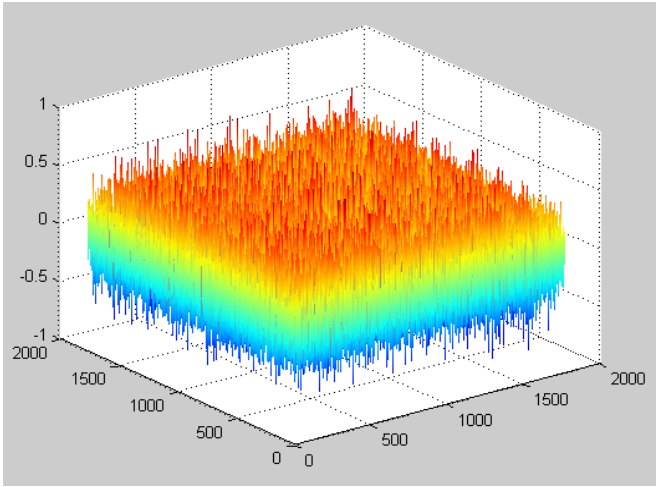
Attacking the Montgomery Ladder

- Attack proceeds in the same manner as before
- However, we are comparing the output of one operation with the input of another
- Given sets of subtraces we compare all the points in one trace with all the points in the other.

- That is, if we assume that a trace taken during the computation of:
  - A doubling operation comprises u_d points, and
  - An addition comprises u_a points.

Attacking the Montgomery Ladder

$$\begin{pmatrix} \bar{c}_{1,1} & \bar{c}_{1,2} & \cdots & \bar{c}_{1,u_a} \\ \bar{c}_{2,1} & \bar{c}_{2,2} & \cdots & \bar{c}_{2,u_a} \\ \vdots & \vdots & \ddots & \vdots \\ \bar{c}_{u_d,1} & \bar{c}_{u_d,2} & \cdots & \bar{c}_{u_d,u_a} \end{pmatrix}$$



- Where each $\bar{c}_{i,j}$ is the i-th point from a doubling operation, for $1 \leq i \leq u_d$, and j-th point from an addition, for $1 \leq j \leq u_a$.

Attacking the Montgomery Ladder

- Construct a matrix assuming a dummy operation always occurs, and another where the random operations are compared
- The difference indicating what combination of points can detect a collision
- Applied to subtraces from individual loops giving correlation coefficients that can be use to determine bits of the scalar as previously

$$D = d_1 \dots d_{n-1} = \{\rho(\bar{a}'_2, \bar{a}'_4), \rho(\bar{a}'_4, \bar{a}'_6), \dots, \rho(\bar{a}'_{m-2}, \bar{a}'_m\}$$

- Repeat for the second observation and take the strongest result.

Success Rates for 192-bit Scalar Multiplication

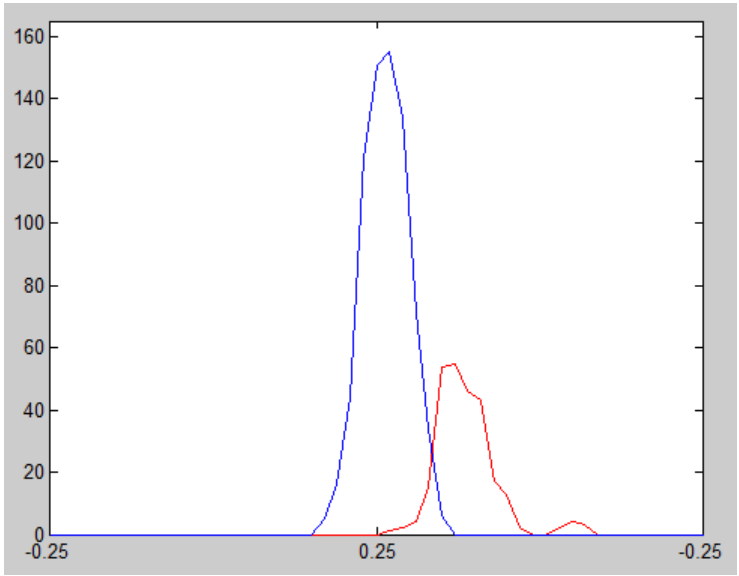
Platform	Algorithm	Matching Method	E (#Errors)	σ	Pr(trivial attack)	Pr(practical attack)
ARM7	Add-only	Euclidean distance	5.78	4.30	0.926	0.991
		Correlation Coefficient	5.52	4.96	0.935	0.993
	Dummy	Euclidean distance	6.10	7.10	0.894	0.968
		Correlation Coefficient	8.40	8.66	0.820	0.920
	Montgomery	Euclidean distance	14.7	4.35	0.306	0.926
		Correlation Coefficient	21.7	4.74	0.0110	0.409
SASEBO	Add-only	Euclidean distance	7.69	2.68	0.955	1
		Correlation Coefficient	24.8	4.93	0.00338	0.190
	Dummy	Euclidean distance	37.7	5.88	0.00188	0.00225
		Correlation Coefficient	24.4	4.88	0.00525	0.207

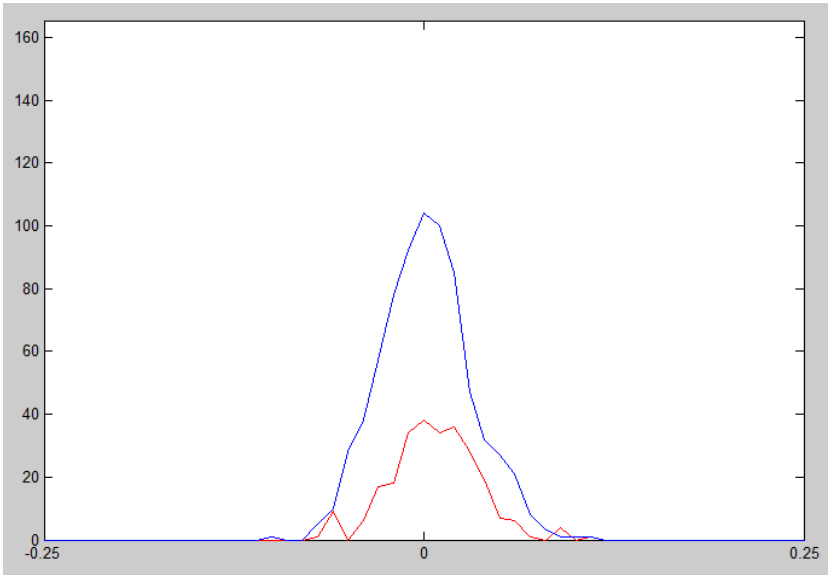
- We define a practical attack as one with time complexity less than 2^{54} (Biryukov et al. 2010)
- A trivial attack is, arbitrarily, set to 2^{40} as requiring modest resources
- Complexity can be derived from algorithms defined by Stinson (2002)

Conclusion

- We demonstrate that collision attacks are a threat using a single trace.
  - Applicable to blinded or ephemeral exponents.
- Given that one can, potentially, observe the use or reuse of variables all algorithms can be attacked.
- However, significant leakage is required for the attack to succeed.
- One can readily test whether an implementation is vulnerable
- Attacks can be prevented by adding noise
  - Randomization of redundant representations
  - Algorithms some random ordering

Example a 4-ary Exponentiation





Cold Boot Attacks in the Discrete Logarithm Setting

B. Poettering [1] & D. L. Sibborn [2]

[1] Ruhr University of Bochum

[2] Royal Holloway, University of London

April, 2015

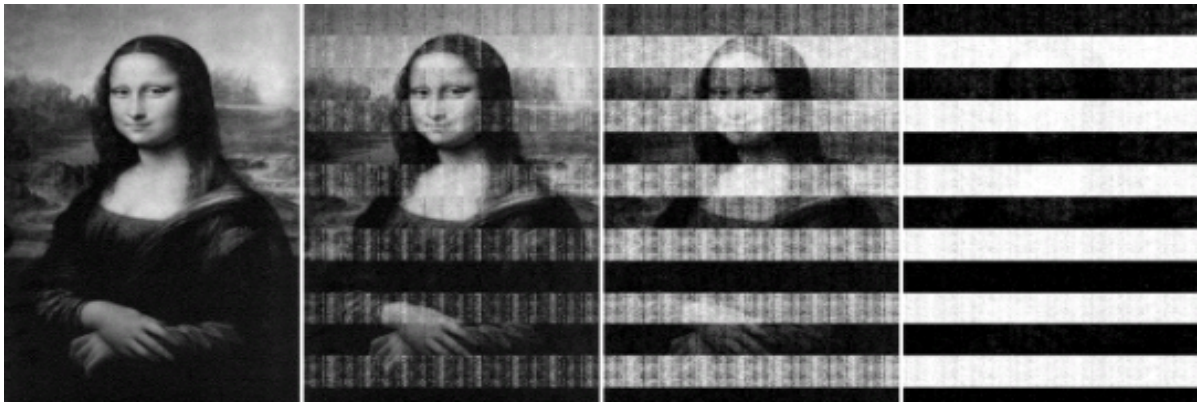
Cold Boot Attacks

- Usenix 2008 - Halderman et al. noted that DRAMs retain their contents for a while after power is lost.
- Bits in memory can be extracted, but they will have errors.
- 0 bits will always flip with very low probability (<1%), but 1 bits will flip with much higher probability which increases with time.
- For example

Original memory: 1100010110110 1001 . . .

Noisy memory: 1110000110010 0001 . . .

Cold Boot Attacks



- Why is this a problem?
- Secrets may be stored in memory.

Important Question

Given a noisy key obtained from a cold boot attack, how can we recover the original key?

Previous Approaches

- This question has been addressed many times before.
- Most cold boot attacks consider the reconstruction of RSA private keys.
- There are attacks against symmetric schemes such as DES and AES.
- There is only one paper that discusses cold boot attacks in the discrete logarithm setting.

Cold Boot Attacks for Discrete Logarithm Keys

- Cold boot attacks usually exploit redundancy in the private key's in-memory representation.
- E.g. in practice RSA private keys contain the parameters $(p, q, d, d_p, d_q, q_p^{-1})$ instead of just d.
- For previous DL cold boot attacks, the authors assumed there was no redundancy in the key.
- If we have redundancy, the previous attacks can be improved.

Important Question

Are there any discrete logarithm implementations that contain redundant information about the private key?

Non-Adjacent Forms (NAFs)

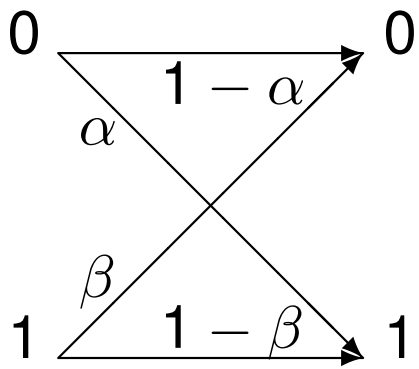
- The simplest NAF re-encodes a scalar $x \in \{0,1\}^{\ell}$ as a string $x' \in \{0,1,-1\}^{\ell+1}$.
- Binary expansion: $7 = 2^2 + 2^1 + 2^0 = 111_2$.
- Alternatively $7 = 2^3 – 2^0$, so $\mathrm{NAF}(111_2) = 1\ \ 0\ \ 0\ \ -1$.
- The NAF is designed to reduce the number of additions.
- For elliptic curves, subtractions are as efficient as additions.
- A modified version of this NAF is used for OpenSSL elliptic curve implementations (called "windowed NAF").
- The NAF is more efficient than the standard double-and-add algorithm.

Comb-Based Methods

- Comb methods are designed to reduce the number of multiplications.
- They require some pre-computation that depends on a fixed base point.
- Basic combs are a re-ordering of the bits.
- PolarSSL employs a modified comb technique.

The Attack Model

- Neither OpenSSL nor PolarSSL explicitly states that the original private key should be discarded.
- Hence, both the original key and its re-encoding (NAF or comb) will be contained in memory, at least for some time.
- We assume an adversary has mounted a cold boot attack and obtains noisy versions of the key and its re-encoding.
- We assume the adversary knows α and β, where bits degrade according to the following channel:



The Reconstruction Technique

- The (textbook) NAF is constructed by starting from the least significant bits.
- i.e., for the simplest NAF, the least t signed digits only rely on knowledge of the least $t+1$ bits of the bit string.
- For example, take the integer 7:

partial bit string	:	partial NAF
$1\ 1$	$\rightarrow$	-1
$1\ 1\ 1$	$\rightarrow$	$0\ -1$
$0\ 1\ 1\ 1$	$\rightarrow$	$0\ 0\ -1$
$0\ 0\ 1\ 1\ 1$	$\rightarrow$	$1\ 0\ 0\ -1$

- Comb encodings have a similar property.

The Reconstruction Technique

- Our reconstruction procedure will consider partial solutions for the private key (across a small section of bits).
- For each candidate we can compute a partial re-encoding (NAF/comb).
- We compare these candidate solutions (and their re-encodings) against the noisy information.
- We keep a (possibly large) list of candidates for which the ‘correlation’ is ‘good’. Candidates with bad correlation are discarded.
- We then consider candidate solutions across a new section of bits, and repeat the procedure.

The Reconstruction Technique (Example for NAFs)

- Suppose we consider 2 bits at a time. We begin like this:

candidate, x	partial-NAF(x)	Correlation
0 0	0	bad
0 1	1	bad
1 0	0	bad
1 1	-1	good

- The second stage would then look like this:

candidate, x	partial-NAF(x)	Correlation
0 0 1 1	1 0 -1	bad
0 1 1 1	0 0 -1	good
1 0 1 1	1 0 -1	bad
1 1 1 1	0 0 -1	good

The Reconstruction Technique

- This process would repeat until the candidate solutions are all of equal size to the private key.
- We can then compare each remaining candidate solution against the public key $Q = aP$.
- If $xP = Q$ for any candidate x, the algorithm outputs x as the private key. Otherwise the algorithm fails.
- A similar technique applies to our comb reconstruction procedure.
- Note, our actual OpenSSL reconstruction differs slightly from the description given here (please see the paper!).

How Do We Measure Correlation?

- How is the correlation measured? However you like!
- We could use Hamming distance, Maximum-Likelihood, . . .
- We could measure the correlation of all bits, or only the newly-added bits, . . .
- BUT, we chose to use a multinomial test because it provides us with a neat theoretical analysis of success.

Multinomial Distributions

- Multinomial distributions are a generalisation of binomial distributions.
- Multinomial distributions have k mutually exclusive events.
- Each of the k events has probability p_i, and $\sum_{i=1}^{k} p_i = 1$.

Multinomial Distributions

- Consider a bowl of sweets from which we sample at random (with replacement):

- Suppose we have four colours, with $\mathbb{P}(\text{red}) = 0.4$, $\mathbb{P}(\text{blue}) = 0.3$, $\mathbb{P}(\text{yellow}) = 0.2$, $\mathbb{P}(\text{green}) = 0.1$.
- If we pick 10 sweets randomly, what is the probability of picking:
  - 5 red, 2 blue, 2 yellow, 1 green?
- The multinomial distribution can tell us the probability of any combination of colours.

Multinomial Test

- Suppose we pick 10 sweets at random and obtain:
  - 0 red, 10 blue, 0 yellow, 0 green.
- Can we be confident that the sweets were chosen from the previous bowl?

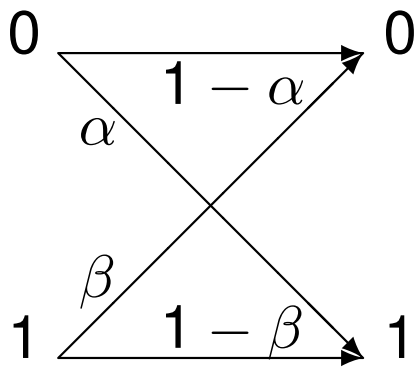
- Maybe the sweets were chosen from another bowl.

Multinomial Test

- The multinomial test is a way of deciding whether a set of observed values is consistent with a particular probability vector (for a specified confidence interval).
- i.e., if we observe 4 red, 3 blue, 2 yellow, 1 green, is it likely that:
  - $\mathbb{P}(\text{red}) = 0.4$, $\mathbb{P}(\text{blue}) = 0.3$, $\mathbb{P}(\text{yellow}) = 0.2$, $\mathbb{P}(\text{green}) = 0.1$?
  - These probabilities seem plausible.
- However, if we observe 0 red, 10 blue, 0 yellow, 0 green, is it likely that:
  - $\mathbb{P}(\text{red}) = 0.4$, $\mathbb{P}(\text{blue}) = 0.3$, $\mathbb{P}(\text{yellow}) = 0.2$, $\mathbb{P}(\text{green}) = 0.1$?
  - These probabilities seem highly unlikely!

Multinomial Test

- How does this help us?
- Recall that our algorithm measures the ‘correlation’ between our candidate key and the noisy bits.
- Recall that in a cold boot attack the bits will degrade according to the following channel:



Multinomial Test

- Hence, there are four possible bit-pairs.
- These are: $0 \to 0$, $0 \to 1$, $1 \to 0$ and $1 \to 1$.
- These four pairs can be viewed as the colours red, blue, green and yellow of the previous example.
- If we let p_b denote the probability of a b-bit appearing in the original key (together with the re-encoding), then:
  - $\mathbb{P}(0 \to 0) = p_0(1 - \alpha)$,
  - $\mathbb{P}(0 \to 1) = p_0\alpha$,
  - $\mathbb{P}(1 \to 0) = p_1\beta$,
  - $\mathbb{P}(1 \to 1) = p_1(1 - \beta)$.

Multinomial Test

- For each candidate solution, we perform a multinomial test.
- If the candidate's degradation is consistent with the probability vector $(p_0(1-\alpha), p_0\alpha, p_1\beta, p_1(1-\beta))$, it is kept.
- Otherwise, the algorithm discards the candidate.
- The user can specify his own confidence interval for the multinomial test.
- This allows the user to recover the private key with an arbitrary success (with a trade-off between running-time).
- N.B. This test also works in the RSA setting (and others!).

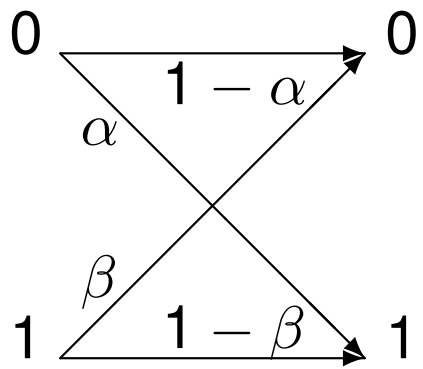
Experiments

- We will shortly see some of our experimental results.
- For each experiment we degraded 100 keys (each of length 160 bits).
- We then used our algorithm to attempt to recover the original keys.

OpenSSL (NAF) Experiments

- For these experiments we set $\alpha = 0.001$. (N.B. There are several extra parameters to the algorithm that are not displayed here.)

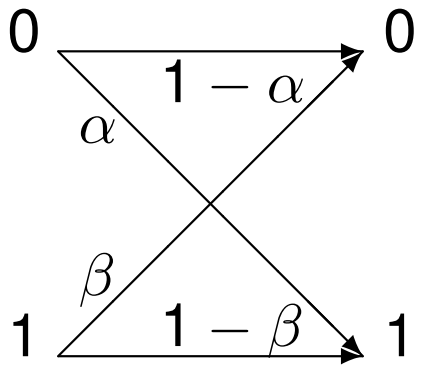
β	0.1	0.15	0.2	0.25	0.3
Predicted Success	0.15	0.15	0.02	0.01	0.01
Success	0.17	0.2	0.07	0.06	0.04



PolarSSL (comb) Experiments

- For these experiments we set $\alpha = 0.001$. (N.B. There are several extra parameters to the algorithm that are not displayed here.)

β	0.01	0.03	0.06	0.08	0.1
Predicted Success	0.73	0.17	0.04	0.01	0.01
Success	0.81	0.6	0.55	0.37	0.08



Predicted Success vs Actual Success

- There is sometimes a big discrepancy between the predicted success and the observed success!
- The predicted success is based on the chi-squared distribution.
- The distribution of the multinomial test converges to the chi-squared distribution.
- For small sample sizes, the convergence is poor.
- Due to the manner of convergence, the chi-squared test provides a lower bound on the multinomial success.

Conclusions

- We have proposed practical key-recovery algorithms against OpenSSL and PolarSSL elliptic curve implementations.
- Our algorithms allow keys to be recovered with a user-chosen success rate (at the expense of running-time).
- The statistical test we use can be implemented with other key-recovery algorithms in other settings, such as RSA.
- Our paper provides the first exposition of the PolarSSL encoding in the cryptographic literature.