Issues of Security with the Oswald-Aigner Exponentiation Algorithm

Colin D Walter

Comodo Research Lab, Bradford, UK

www.comodogroup.com

Overview

- Side Channel Leakage
- History
- The Oswald-Aigner Exponentiation Algorithm
- Recovering Secret Key Bits
- Counter-measures
- Conclusion

Side Channel Leakage

- Gates use of power is state and data dependent.
- Wire transmission of power is data dependent.
  - So current & EMR are data dependent.
  - For example, noticeable differences between loading data and commencing a long integer computation.
- Conditional Statements are data driven.
  - So execution time may be data dependent
  - For example, a conditional modular subtraction may have only one arm. So the value of the condition may be deducible.
- Conclusion: secret key information may leak.

History

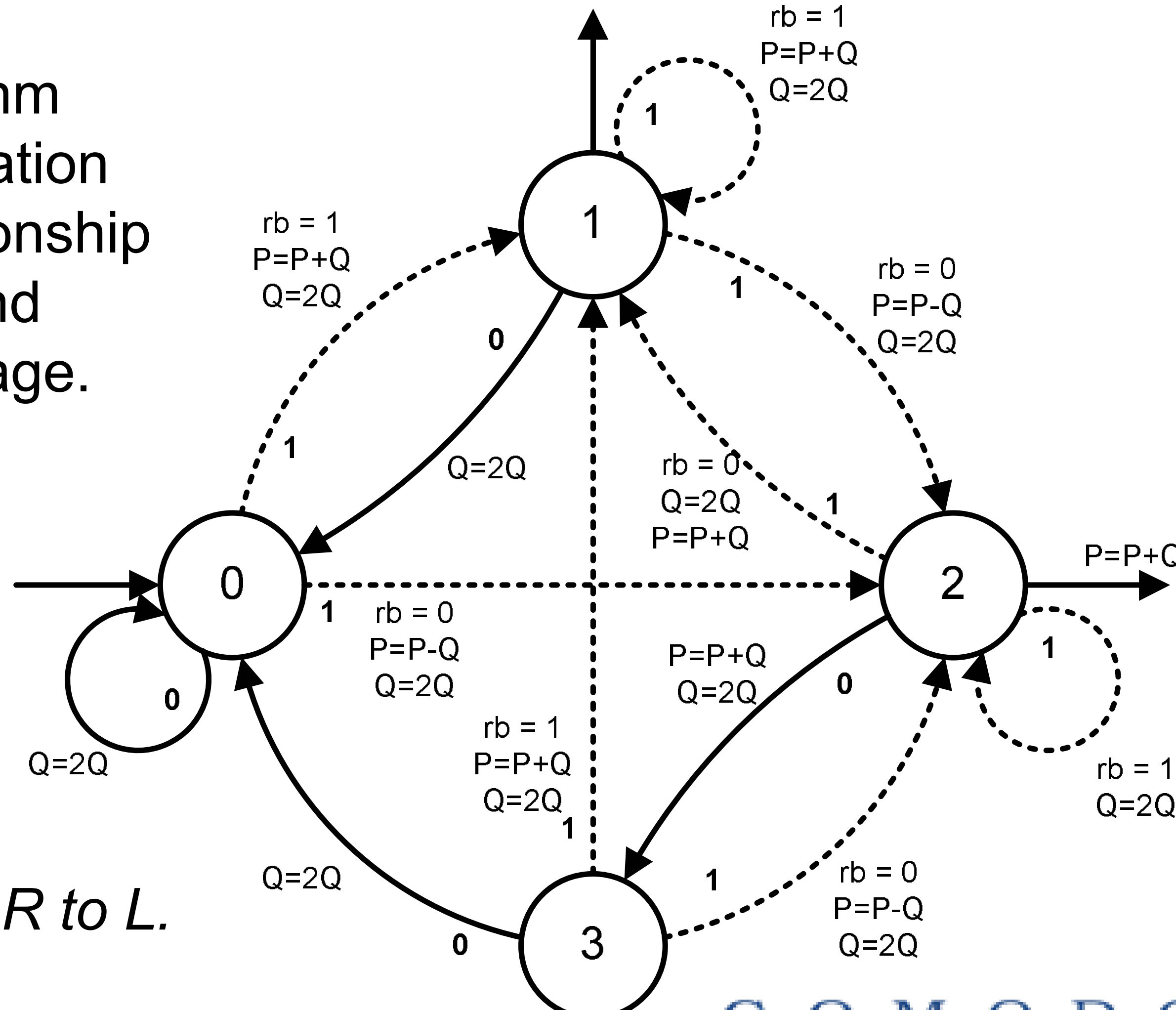
- NSA – *no such activity?* – Tempest shielding.
- Kocher *et al* (1996-7): Timing & Power side channel papers.
- Walter & Thompson (2001): Theory for practical attack on RSA.
- Oswald, Aigner, Smart, Liardet (2001): Randomised Algorithms.
- Walter (2002): Liardet-Smart – use unblinded keys only once.
- Okeya & Sakurai (2002): Oswald-Aigner, special case.
- Here (2004): Oswald-Aigner, extended general case.

The Oswald-Aigner Algorithm

The Exp^n Algorithm contains randomisation to obscure the relationship between data and side channel leakage.

Finite Automaton to compute $P = kQ$

rb = random bit
secret key k: read R to L.



RSA Conference 2004

C·O·M·O·D·O
RESEARCH LAB

Main Assumptions

Suppose:

1. Doubles & Adds can be distinguished using power/EMR/time.
2. Adds & Subtracts are indistinguishable.
3. The secret *k* is re-used many times (≥10, say) without blinding.
4. The random bits *rb* are chosen independently
   with fixed probability which depends on the current state
   of the automaton.
5. *k* has uniformly distributed, independently chosen bits, ...

Recovering Secret Key Bits (1)

- Each point multiplication generates a word over **{*D*,*A*}** where the number of ***D***s is the number of bits in ***k***.
  - e.g. **11001** yields **AD D D AD AD** under r-to-l binary exp^{n} alg^{m}

    Here other choices are also possible.
- The i^{th} "***D***" is generated by the i^{th} bit of k, so we can align traces.
- Patterns **AD**, **D** and **DA** are possible for a bit.
- Between two ***D***s: **DD**, **DAD** and **DAAD** are possible.
- The relative frequencies determine the bit of k.

Recovering Secret Key Bits (2)

Example Strings Key **11001**, ***D***s aligned:

	1		**0**		**0**		**1**		**1**		←bit order reversed
A	D		D		D	A	D	A	D		←binary exp case
A	D		D		D	A	D	A	D	A	
A	D		D		D	A	D		D	A	
A	D		D		D	A	D		D	A	
A	D	A	D		D	A	D	A	D		
A	D	A	D		D	A	D	A	D	A	
A	D	A	D		D	A	D		D	A	
A	D	A	D		D	A	D		D	A	

Recovering Secret Key Bits (3)

1. For each bit pattern, compute frequency of each {A,D} pattern.
2. Deduce the possible bit patterns, ranked by likelihood.
3. Remove inconsistencies where associated bit strings overlap.

- Observation: it is easier to recognise bits from longer patterns.

Recovering Secret Key Bits (4)

If $p(rb=1) = \frac{1}{2}$ in each state, then the prob of each state is:

0: ⅜	1: ¼
2: ¼	3: ⅛

So the prob of each *D*-to-*D* sub-string is:

DD	15/32
DAD	16/32
DAAD	1/32

Recovering Secret Key Bits (5)

- *Example*. 2 bits is enough with ~10 traces. For bit pair $k_{i+1}k_i$:

  DAAD in some trace means it *is* 11,
  DAD or DAAD in some trace means it is *not* 00,
  no DAD or DAAD means 00 (*probably*),
  no DD or DAAD means 10 (*probably*),
  both DD and DAD means it *is* x1,
  both DD and DAD but no DAAD means 01 (*probably*).

- *Apply this* to example on slide *8*.

  (Above bit order as in key, but reversed in trace table.)

Deduction Errors

- Using 10 traces to deduce the most likely bit pair and assuming $p(rb) = ½$, ***only 1 in 166 bits is incorrect.***
- It is ***computationally feasible*** to search for, and recover, k with standard ECC key length.
- Precisely, k is recovered from $O(\log \log k)$ traces
  and $O((\log k)^2)$ decryptions
- Bits are recovered from *local* data, not sequentially L-to-R or R-to-L, so less re-computation when errors are made.
- Clearly, therefore, this algorithm should ***not*** be used where the initial assumptions hold.

Counter-Measures

- Can the parameters be changed to improve security?
- *No!* Whatever the chosen probability of *rb* = 1 in a given state, similar deductions can be made.

Solutions:

1. Add fresh, random blinding for each use of *k*.
2. “Add and *always* double”, so every DD, DAD and DAAD is disguised as *DAAD* (very expensive).
3. “Balanced” code which is the same for A and D.

Dangers

- Longer {A,D}-patterns are more exclusive: the under-lying {0,1}-pattern may have uniquely determined substrings.
- Fewer traces but more computation are required with this approach.
- Experimentally, $O(\sqrt[4]{k})$ keys match a pattern given by k.
- In fact, about 20 patterns for a 16-bit section of a key, and hence $20^{16} \approx 2^{69}$ decryption checks for a 256-bit key. This is just computationally feasible.
- So a ***single*** use of the key with this algorithm may be unsafe (making key blinding insufficient as a counter-measure).

Alternative Randomised Algorithms

- Besides the previous counter-measures, there are more secure randomised algorithms:

1. MIST: Walter (RSA 2001)
2. Overlapping Windows: Itoh *et al* (CHES 2002)

Conclusions

- The original randomised algorithm of Oswald & Aigner can only be used securely for a few times with the same key unless other counter-measures are employed (although it is undoubtedly more secure than "square and multiply".)
- No parameter choice improves the situation.
- Standard counter-measures improve the security.
- The analysis is applicable to other randomised algorithms where at each point the unprocessed part of the key is fixed.
- It is clearer how to construct safer randomised algorithms.
- There are also suitable alternative algorithms.