Improving Smart Card Security using Self-timed Circuits

Simon Moore, Ross Anderson, Paul Cunningham, Robert Mullins, George Taylor
Computer Laboratory, University of Cambridge
simon.moore@cl.cam.ac.uk

Abstract

We demonstrate how 1-of-n encoded speed-independent circuits provide a good framework for constructing smart card functions that are resistant to side channel attacks and fault injection. A novel alarm propagation technique is also introduced. These techniques have been used to produce a prototype smart card chip: a 16-bit secure processor with Montgomery modular exponentiator and smart card UART.

1. Introduction

Smart cards are increasingly prevalent, particularly in Europe, for authentication and payment mechanisms (credit cards, pay-TV access control, public transport payment, medical records, personal identity, mobile phone SIMs, etc.). They present a harder target for the criminal underworld than their magnetic strip counterparts. None the less, there is sufficient economic gain in cracking smart cards. Pay-TV is particularly vulnerable since communication with the smart card is typically unidirectional, from the broadcasting source to the set-top box hosting the smart card. Since there is no back channel, it is not possible to identify duplicate smart cards via interactive protocols. Consequently, it is economically attractive to reverse engineer a pay-TV smart card in order to make a large number of duplicates. As smart cards are used in more and more applications, many new opportunities for theft and fraud open up to criminals capable of reverse engineering cards or extracting key material.

The next section introduces attack technologies which determine the environment in which smart cards must survive. We address a number of hardware level security issues and how self-timed circuits can be used to build more robust smart cards.

2. Attack Technologies

Hardware level attacks fall into two main categories: invasive and non-invasive attacks.

2..1 Invasive attacks

Reverse engineering is the most extreme form of an invasive attack where the smart card is depackaged and completely analysed. Monitoring of bus signals is often sufficient to extract data, and can be undertaken by dropping picoprobes on bus lines. If bus signals are hidden (e.g. by a top level metal defence grid), a focused ion beam (FIB) workstation may be used to extract signals. There is also the 'litigation attack'; the attacker first obtains a patent that might possibly have been infringed by a smart card designer, then abuse the legal discovery process to obtain design details. Thus, inline with Kerckhoffs' principle[1], one has to assume that the design details of a smart card are in the public domain.

Another attack technique, used in the context of an invasive microprobing attack, is to use a laser to shoot away alarm circuitry, or protective circuitry such as access control matrices which allow certain areas of memory to be accessed only after the presentation of certain passwords [1].

2..2 Non-invasive attacks

More recently non-invasive attacks (sometimes called *side channel attacks*) have been investigated. An early approach was to measure data dependent timing in order to extract key information [2]. Another approach is differential power analysis [3] where keys are extracted from the differences between runs of data dependent power emissions. A variation of this attack extracts keys via electromagnetic analysis (EMA). The use of $40\mu m$ antennas can easily pin-point individual power distribution nets and can be used to build up a profile of on chip power consumption [4] (see Figure 1).

Non-invasive attacks are of particular concern because they leave no evidence of tampering can be undertaken relatively quickly. Software techniques are used to introduce randomness to the execution process to make side channel attacks more difficult. However, to reduce data dependent power leakage requires careful circuit

[1]Kerckhoffs' principle: "The security of a cryptosystem must not depend on keeping secret the crypto-algorithm. The security depends only on keeping secret the key." translated from *La cryptographie militaire* (1883).

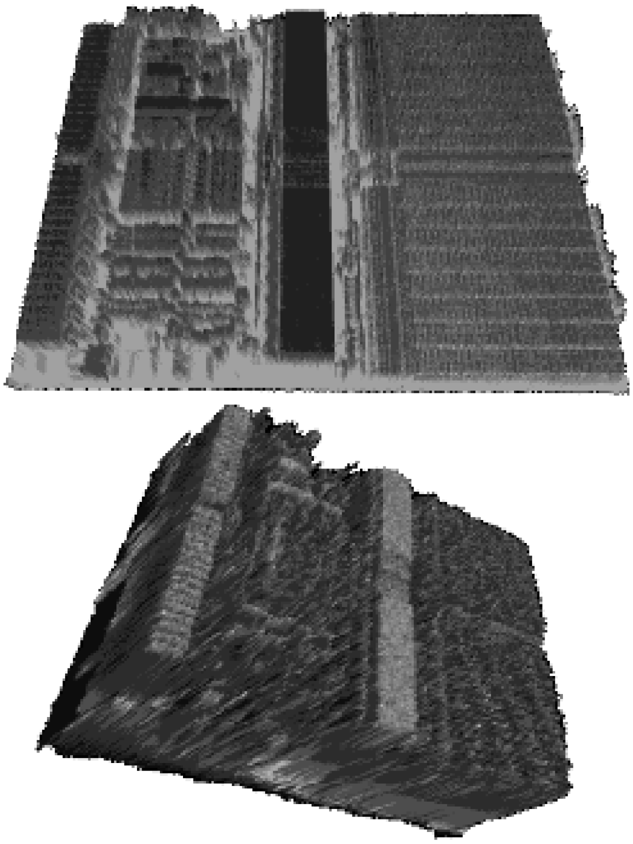

Figure 1: example of an EMA chip scan

design.

Another threat to smart card systems is fault induction. Faults are induced in a number of ways, such as by introducing transients ('glitches') on the power and clock lines [5, 6] and illuminating one or more transistors with a laser. These can cause the processor to malfunction in a predictable and instructive way. For example, when a processor executes a branch instruction, the write to the program counter is often conditionally set late in the clock cycle. An abnormally high clock frequency may result in branches not being taken. Skipping a branch instruction can result in a jump to some failure case being nulled, thereby bypassing security checks.

This paper describes techniques for improving smart card security by:

- eliminating data dependent power consumption
- defending against fault induction by using a redundant encoding scheme
- diffusing data dependent timing
- propagating alarm signals as an integral part of the data

Our emphasis is on making non-invasive attacks impractical because this class of attack poses the greatest economic threat.

3. Eliminating data dependent power consumption

Simple binary encoding of data, where one wire is used to propagate one bit, results in power consumption proportional to the number of state changes. Data transmission along a bus consumes considerable power due to wire capacitance. Bus activity is observed as the Hamming weight of the state changes.

One approach to reducing data dependent power consumption is to use an alternative data encoding scheme. For example, one-hot data encoding (1-of-n codes) consume constant power to transmit data since just one wire transitions for every symbol. 1-of-2 (dual-rail) and 1-of-4 data encodings are commonly used in self-timed circuit design [7, 8].

1-of-n encoding is not sufficient to guarantee a data independent power signature. Firstly, the path taken by each wire is likely to vary, which can result in a difference in wire load. This problem may be addressed by careful layout of long buses and careful floor planning to keep random wire lengths under control.

A second problem is the logic complexity variation between each wire. Care has to be taken when choosing standard cells to minimise this effect. Often this constraint means that silicon area has to be sacrificed for greater security. However, most smart card chips are dominated by memory requirements so additional area for logic adds little to the final size of the chip.

Data dependent control presents a third problem. For example, a hardware multiplier implemented using a shift-and-conditional-add algorithm uses data-dependent power if the add operation is only undertaken when required. A similar problem occurs if early completion detection is used. Ensuring that the same operations occur regardless of the data seems to be the best way to deal with this problem. An alternative approach is to add random noise to the power signature, but randomness can often be removed by signal averaging over repeated runs.

4. Defending against fault induction

Self-timed designs are immune to clock glitch attacks. Where the clock is required, for example in the clocked serial smart card interface, it is easy to arrange that data corruption should not result in sensitive data leakage.

A speed independent (SI) asynchronous circuit naturally adapts to power supply voltage, thereby making power supply glitch attacks less successful. However, self-timed circuits cannot protect components like EEPROM which is often used to store keys and error counters.

Dual-rail encoding is often used to construct SI circuits. Two wires are used to encode three states: *clear, logic-0 and logic-1* (see Figure 2). The *unused* fourth state (often 11_2) is typically not used by dual-rail circuits. However, from the stand point of managing faults, the unused state must be explicitly handled as an error condition which we shall call *alarm* (more of how the *alarm* is used appears in the next section).

A1	A0	meaning
0	0	clear
0	1	logical 0
1	0	logical 1
1	1	alarm

Figure 2: dual-rail encoding with alarm signal defined

Single point fault induction results in one of three behaviours:

1. Data can be suppressed which results in the circuit deadlocking in the *clear* state.
2. A *clear* state is forced into a *logic-0* or *logic-1* state which typically results in deadlock in the control path because an extra data item has miraculously appeared. Deadlock under these circumstances can be guaranteed.
3. A logic state is forced into the *alarm* state resulting in data corruption. The *alarm* signal can be propagated rapidly resulting in data being deleted and a global alarm raised.

The fact that SI circuits deadlock when a fault occurs is very useful because it paralyses the chip until a hard reset is performed. This is in contrast to clocked circuits where induced faults are in the same league as pulses due to static and dynamic hazards, which are considered normal operating behaviour in many synchronous designs.

5. Alarm propagation and detection

There are many possible tamper sensors, including over/under voltage, temperature and light level sensors (to detect depackaging). All emit a signal when the appropriate condition is violated. The *alarm* state, as indicated in the previous section, can be activated by ORing an *alarm* control signal from the tamper sensor with the dual-rail data path (see an example fragment of data-path in Figure 3). The *alarm* state is detected by an AND–OR tree on the data. The amount of circuitry is substantially reduced when it is known that setting the alarm for one logical bit will result in the alarm being propagated to other bits — see the worked example in Section 7.

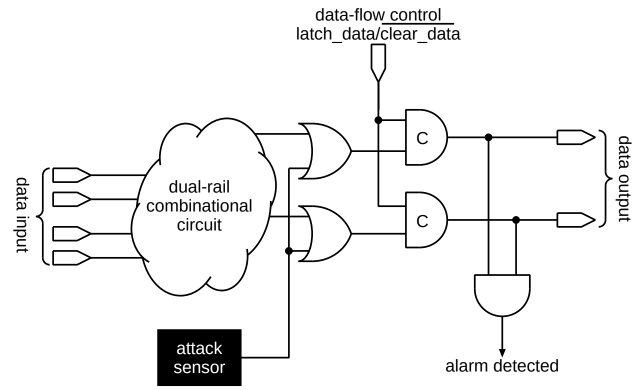

Figure 3: Alarm insertion and detection

6. Diffusing data dependent timing

Timing attacks on clocked processors are undertaken by counting cycles between known events in order to derive secret information [2]. An asynchronous processor is more susceptible to leaking data in the time domain. For example, an adder could be carefully designed to consume constant energy per operation, but the duration of the operation is dependent upon the propagation of the carry. In this instance the adder must be designed to always achieve worst case performance. This does not mean that the adder must have bad performance. For example, we have designed a SI carry select adder which operates quickly and with constant timing properties.

Variation in wire and gate loads can also lead to data dependent timing. Wire loads can be minimised by applying suitable constraints during place and route. If wire loads are to be made insignificant then the design is partitioned into a number of small blocks. Wire load balancing then only needs to be considered at interface boundaries.

Gate loads are managed by using constant transistor sizing. Fan-out in SI circuits is typically symmetric for dual-rail wires, since as many conditions need to be tested if the value is a *logical-0* as they do when the value is a *logical-1*. For example, see the full adder in Section 7.

Where there is concern over data dependent timing, random delays may be inserted using the circuit in Figure 4 which is based upon a Seitz arbiter [9].

Data dependent cycling is also an issue. For example, an iterative multiply might terminate early when only zero bits

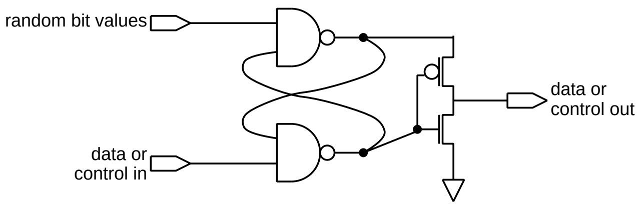

Figure 4: Random delay insertion

are left in the multiplicand. The multiply could be detuned if necessary. However, the fact that a multiply is being performed can often be detected, since the instruction fetch might be stalled and there may be no data memory activity (depending upon the CPU architecture of course). This is a significant problem since memory accesses tend to consume considerable power and so memory access patterns show up easily. However, the attacker's life can be made harder by inserting dummy memory accesses in situations where the memory interface would otherwise be idle.

Data dependent software timing is a further issue. It is imperative that application software for a smart card works in collaboration with the hardware. One approach is to ensure that the same work is undertaken regardless of the data. However, this soon becomes difficult and results in very slow code. An alternative approach is to introduce nondeterminism at the software level (see the example in Figure 5). Dynamic instruction scheduling in hardware to produce nondeterministic instruction execution orders is also possible [10, 11]. Instruction level nondeterminism adds considerable complexity to an analysis of side channel information from repeated runs. However, it is important to note that this form of nondeterminism only adds complexity if the side channel information is hard to analyse in the first place.

```
if(random_condition) {
            f(); g();
} else {
            g(); f();
}
```

where `f()` and `g()` are independent functions

Figure 5: Adding nondeterminism to software

7. Worked Example

This section describes the design of a dual-rail OR gate and a dual-rail full-adder. To improve security data-independent timing and power consumption is sought. We also wish to be able to maximise propagation of *alarm* signals.

A simple dual-rail OR gate is show in Figure 6. If an *alarm* signal is present on (a_1, a_0) or (b_1, b_0) inputs then the output r_1 will always be set. However, the r_0 signal will not be set if only one of the dual-rail inputs is at *alarm*. Also, the OR gate takes a data dependent time to produce and output. These two issues are resolved in Figure 7 by adding additional logic to detect the alarm state, forcing r_0 high and at the same time making the delay data independent. The C-elements in the alarm propagation circuit could be replaced with AND gates if one was confident that the alarm signal coming in was stable. However, under a glitch attack we cannot be confident of this so the storage aspect of the C-elements is desirable.

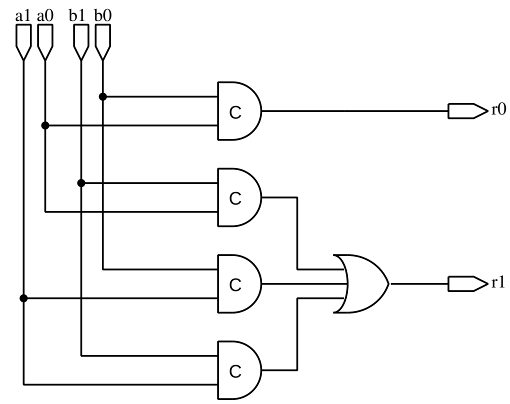

Figure 6: dual-rail SI OR gate

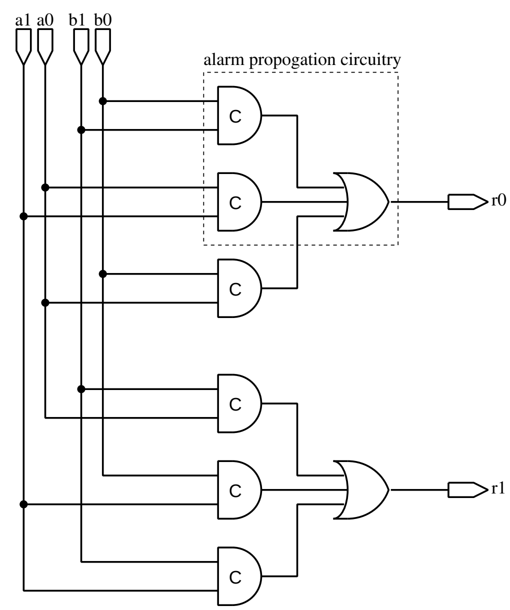

Figure 7: dual-rail SI OR gate with alarm propagation

The OR gate example illustrates how path delays can be balanced. However, this is not always necessary. For example, let us consider a dual-rail full-adder. The dual-rail inputs are (a_1, a_0), (b_1, b_0), (cin_1, cin_0) and the outputs are (sum_1, sum_0) and $(cout_1, cout_0)$. Since we require several functions of the inputs it is worth expanding the inputs into a 1-of-8 code vis:

$$
\begin{aligned}
i000 &= C(a_0, b_0, cin_0)\\
i001 &= C(a_0, b_0, cin_1)\\
i010 &= C(a_0, b_1, cin_0)\\
&\vdots\\
i110 &= C(a_1, b_1, cin_0)\\
i111 &= C(a_1, b_1, cin_1)
\end{aligned}
$$

where $C()$ represents a C-element function

Now we can define the outputs as an OR-plane of the 1-of-8 code:

$$
\begin{aligned}
sum_0 &= i000 + i011 + i101 + i110\\
sum_1 &= i001 + i010 + i100 + i111\\
cout_0 &= i000 + i001 + i010 + i100\\
cout_1 &= i011 + i101 + i110 + i111
\end{aligned}
$$

This approach results in data independent timing but the *alarm* signal is not guaranteed to propagate. The *alarm* signal may be identified and added to there sum and $cout$ terms:

$$
\begin{aligned}
alarm &= RS(globalReset, a_0.a_1 + b_0.b_1 + cin_0.cin_1)\\
sum_0 &= i000 + i011 + i101 + i110 + alarm\\
sum_1 &= i001 + i010 + i100 + i111 + alarm\\
cout_0 &= i000 + i001 + i010 + i100 + alarm\\
cout_1 &= i011 + i101 + i110 + i111 + alarm
\end{aligned}
$$

where $RS()$ is an RS flip-flop (reset dominant)

The SR flip-flop is used to keep $alarm$ high causing deadlock. Many dual-rail functions may be implemented in this SI form: first expand the inputs to a 1-of-n code and then have an OR plane to determine the outputs.

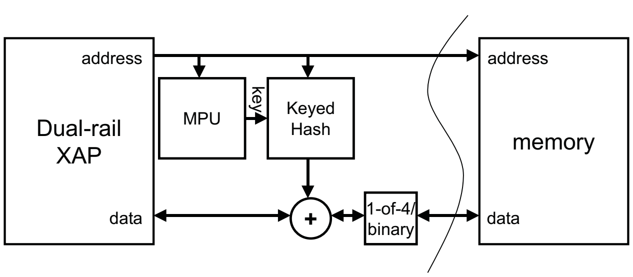

Figure 8: Bus cryptography architecture

8. Case Study: Bus Cryptography

It is convenient to be able to use off the shelf memory components when designing a secure system. Such parts typically have a conventional binary encoded interface. Consequently these devices, and the buses going to them, consume data dependent power. This may be ameliorated in two ways:

1. balance the power consumption
2. encrypt the data

Balancing the power may be achieved in a number of ways. One approach is to write the true and complement of the data at the same time. However, this doubles the amount of storage required which is far from ideal. Power consumption during writes may be balanced using dummy circuitry, but this is not possible with reads since you do not know what it is going to read, and hence what power needs balancing, until the read operation has been performed.

An alternative strategy is to encrypt the data held in the memory. We have developed the scheme depicted in Figure 8. A dual-rail 16-bit processor (called a XAP) has a memory protection unit which supplies a 64-bit key per protection region. The hash of the key and address is used to XOR with the data. On writes, the hash function is on the critical path, but this performance impact can be amortised by pipelining. On reads, the hash is computed in parallel with the memory access. In our system, computation of the hash is fractionally quicker than the memory access so there is no performance degradation.

In our prototype chip, the hash is computed in four rounds (in a production chip with a 32-bit bus, we would have six rounds for cryptographic security). Each round consists of an XOR of the input with part of the key followed by a set of 4-input to 4-output S-boxes taken from the Serpent block cypher [12]. There is then a further layer of XOR. In our system the hash is computed using a 1-of-4 encoded data. We compute the XOR by extracting all 16 combinations of the two 1-of-4 inputs using 16 C-elements. The 1-of-4 result is then computed by an OR plane (in Figure 11 the C-element plane is inverting so a NAND plane is then used). Next the S-boxes and transposition have to be implemented. The S-boxes are a complex function of the 4 logical bits of input. The two pairs of 1-of-4 inputs are first converted into a 1-of-16 code. Each S-box and transposition is then just a transposition of the 1-of-16 code, giving a 1-of-16 output. Given that the final function required is a large XOR, having the data in a 1-of-16 form is advantageous. As a result, the S-boxes and transposition functions are reduced to wiring which is almost for free. Thus, the computation is dominated by the XOR functions. In our implementation, four rounds of the keyed hash function takes 3.3ns which is comparable to the 3.5ns access time for the low power SRAMs used for data storage (on a 0.18μm CMOS process). The 1-of-4 XOR functions are perfectly balanced and propagate the *alarm* signal without further modification.

While designing this unit, we observed a vulnerability of some existing bus cryptography systems. This is that multiple writes of the same data to the same address yield the same enciphered data, as the key in such systems remains unchanged. This means that coincidences in data become apparent to an observer. With some algorithms this is highly undesirable. For example, if the chip software is implementing a common block cipher such as the Data Encryption Standard, the observation of repeated identical values of round keys may enable the cipher key in use to be deduced with high probability. The solution we implemented is to design the memory protection unit (with which the bus cryptography unit is integrated) so that it enables the operating system to treat segments of memory as write-once. Thus each round of a block cipher can use a different segment of memory, encrypted with a key that is cleared after use.

9. Test Chip

We have recently received a test chip with five 16-bit XAP processors, a Montgomery modular exponentiator (for fast public key exchange using RSA with keys of up to 2048 bits), an I/O block which includes a smart card UART, and a distributed 1-of-4 interconnect (see Figure 9). One of the 16-bit processors has been specifically designed with security in mind and has a memory protection unit with bus cryptography attached.

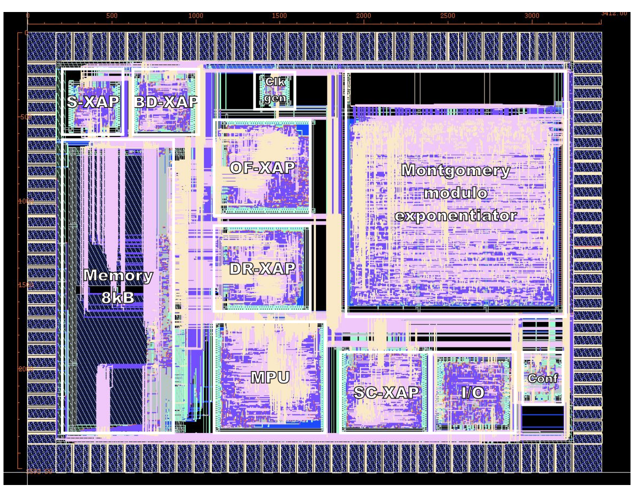

Figure 9: Test chip

10. Conclusions

Until now, the defence mechanisms reported against side channel attacks have been ad-hoc and limited, including clock jittering and random noise generation at the hardware level, and various masking techniques at the software level. None of these has been particularly satisfactory on its own, and using them in combination imposes serious dependencies and costs during both the development and evaluation of systems. There have been no satisfactory chip-level defence mechanisms at all against attackers who induce transient faults using lasers.

This paper discusses many issues in the design of better smart cards. We describe how 1-of-n data encoded SI circuits may be used to make smart cards more resistant to:

- *Timing analysis* by minimising data dependent gate delays and adding random delay to mask residual emissions.
- *Power and electromagnetic emissions analysis* by ensuring that power consumption is data independent.
- *Clock and power glitch attacks* since SI circuits are independent of the clock and adapt to power supply variation.
- *Fault injection* since deadlock or alarm propagation will result.

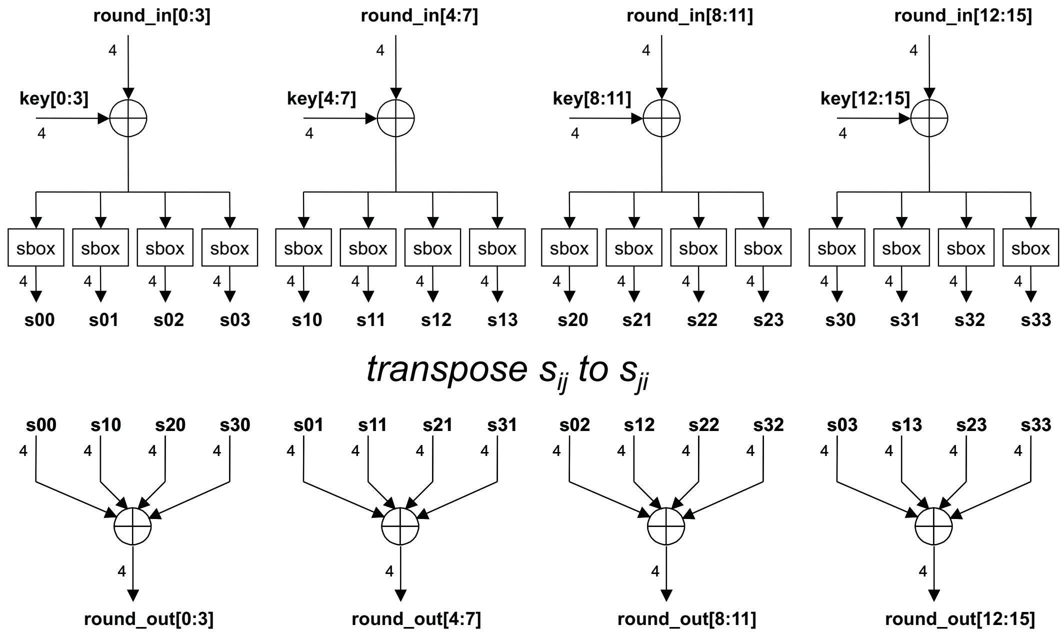

Figure 10: Overview of the keyed hash function

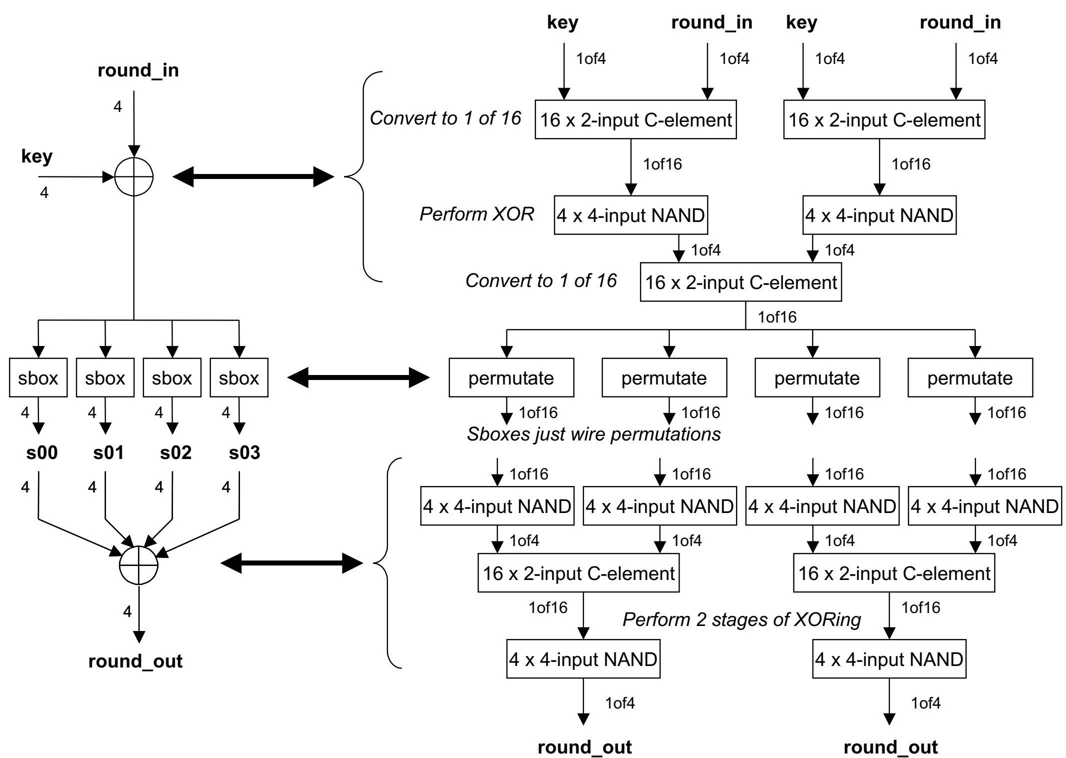

Figure 11: Implementation of the keyed hash function

References

[1] O. Kömmerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors," in *First USENIX Workshop on Smartcard Technology*, (Chicago, Il), pp. 9–20, USENIX, May 1999.

[2] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems," in *Proc. 16th International Advances in Cryptology Conference – CRYPTO '96*, pp. 104–113, 1996.

[3] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. 19th International Advances in Cryptology Conference – CRYPTO '99*, pp. 388–397, 1999.

[4] J. J. Quisquater and D. Samyde, "ElecroMagnetic analysis EMA: Measures and coutermeasures for smart cards," in *Smart Card Programming and Security*, pp. 200–210, LNCS2140, 2001.

[5] E. Sprunk, "Clock frequency modulation for secure microprocessors." US Patent 5404402, Filed 21 December, 1993.

[6] R. Anderson and M. Kuhn, "Tamper resistance – a cautionary note," in *Second USENIX Workshop on Electronic Commerce*, (Oakland, California), pp. 1–11, USENIX, Nov. 1996.

[7] C. Myers, *Asynchronous Circuit Design*. John Wiley and Sons, July 2001.

[8] T. Verhoeff, "Delay-insensitive codes - an overview," *Distributed Computing*, vol. 3, no. 1, pp. 1–8, 1988.

[9] C. L. Seitz, "System timing," in *Introduction to VLSI Systems* (C. A. Mead and L. Conway, eds.), ch. 7, Addison-Wesley, 1980.

[10] D. May, H. L. Muller, and N. P. Smart, "Non-deterministic processors," in *Information Security and Privacy, LNCS 2119* (V. Varadharajan and Y. Mu, eds.), pp. 115–129, Springer Verlag, July 2001.

[11] D. May, H. L. Muller, and N. P. Smart, "Random register renaming to foil DPA," in *Cryptographic Hardware and Embedded Systems CHES 2001, LNCS 2162* (C. K. Koc, D. Naccache, and C. Paar, eds.), pp. 28–38, Springer Verlag, May 2001.

[12] R. J. Anderson, E. Biham, and L. Knudsen, "Serpent: A proposal for the Advnaced Encryption Standard," 2001. at http://www.cl.cam.ac.uk/ rja14/serpent.html.

Acknowledgements

The authors would like to thank Vincent Rijmen for his help with the bus cryptography scheme. Sergei Skorobogatov and Markus Kuhn provided useful insights into smart card attack techniques. Thanks are also due to David Samyde for discussions about and photographs of electromagnetic emissions analysis. We are also grateful to Scott Fairbanks for his comments. We thank the European Commission for funding some of this research as part of the project: IST-1999-13515.