Changing of the Guards: a simple and efficient method for achieving uniformity in threshold sharing

Joan Daemen[1,2]

[1] Radboud University
[2] STMicroelectronics

Abstract. Since they were first proposed as a countermeasure against differential power analysis (DPA) in 2006, threshold schemes have attracted a lot of attention from the community concentrating on cryptographic implementations. What makes threshold schemes so attractive from an academic point of view is that they come with an information-theoretic proof of resistance against a specific subset of side-channel attacks: first-order DPA. From an industrial point of view they are attractive as a careful threshold implementation forces adversaries to DPA of higher order, with all its problems such a noise amplification. A threshold scheme that offers the mentioned provable security must exhibit three properties: correctness, incompleteness and uniformity. A threshold scheme becomes more expensive with the number of shares that must be implemented and the required number of shares is lower bound by the algebraic degree of the function being shared plus 1. Defining a correct and incomplete sharing of a function of degree d in $d+1$ shares is straightforward. However, up to now there is no generic method to achieve uniformity and finding uniform sharings of degree-d functions with $d+1$ shares has been an active research area. In this paper we present a simple and relatively cheap method to find a correct, incomplete and uniform $d+1$-share threshold scheme of any S-box layer consisting of degree-d invertible S-boxes. The uniformity is not implemented in the sharings of the individual S-boxes but rather at the S-box layer level by the use of feedforward and some expansion of shares. When applied to the Keccak-p nonlinear step χ, its cost is very small.

Keywords: side-channel attacks, threshold schemes, uniformity, Keccak

1 Introduction

Systems such as digital rights management (DRM) or banking cards try to offer protection against adversaries that have physical access to platforms performing cryptographic computations, allowing them to measure computation time, power consumption or electromagnetic radiation. Adversaries can use this side channel information to retrieve cryptographic keys. A particularly powerful attack against implementations of cryptographic algorithms is differential power analysis (DPA) [7]. This attack can exploit even the weakest dependence of the power consumption (or electromagnetic radiation) on the value of the manipulated data by combining the measurements of many computations to improve the signal-to-noise ratio. The simplest form of DPA is first-order DPA, that exploits the correlation between the data and the power consumption. To make side channel attacks impractical, system builders implement countermeasures, often multiple at the same time.

In threshold schemes [10,8,9] one represents each sensitive variable by a number of shares (typically denoted by $d+1$) such that their (usually) bitwise sum equals that variable. These shares are initially generated in such a way that any subset of d shares gives no information about the sensitive variable. Functions (S-boxes, mixing layers, round functions …) are computed on the shares of the inputs resulting in the output as a number of shares. An essential property of a threshold implementation of a function is that each output share is computed from at most d input shares. This is called *incompleteness* and guarantees that that computation cannot leak information about sensitive variables. The resulting output is then typically subject to some further computation,

again in the form of separate, incomplete, computation on shares. For these subsequent computations to not leak information about the sensitive variables, the output of the previous stage must still be uniform. Therefore, in an iterative cryptographic primitive such as a block cipher, we need a threshold implementation of the round function that yields a uniformly shared output if its input is uniformly shared. This property of the threshold implementation is called *uniformity*.

Threshold schemes form a good protection mechanism against DPA. In particular, using it allows building cryptographic hardware that is guaranteed to be unattackable with first-order DPA, assuming certain leakage models of the cryptographic hardware at hand and for a plausible definition of "first order".

Constructing an incomplete threshold implementation of a non-linear function is rather straightforward. To offer resistance against first-order DPA, the number of shares equals the algebraic degree of the function plus one. However, constructing one that is at the same time incomplete and uniform is perceived as a challenge. For instance, for the KECCAK S-box, incomplete 3-share threshold implementations are easy to generate but no uniform one is known. Exhaustive investigations have been performed on all small S-boxes (3 to 5 bits) and there are many S-boxes for which it is not known to build uniform threshold implementations with $d+1$ shares if their algebraic degree is d.

Given a non-uniform threshold implementation, it is not immediate how to exploit its non-uniformity in an attack. A paper that makes a start in explorations in that direction is [6]. However, uniformity of a threshold implementation is essential in its information-theoretical proof of resistance against first-order DPA. In short, if one has a uniform sharing, one does not have to give additional arguments why the implementation would be secure against first-order DPA.

In this paper we present a simple and efficient technique for building a threshold implementation with $d+1$ shares of any invertible S-box layer of degree d that is correct, incomplete and uniform. When applied to the nonlinear layer in KECCAK, χ, it can be seen as the next logical step of the methods discussed in Section 3 of [5].

1.1 The *Changing of the Guards* idea in a nutshell

The basic method can be summarized as follows:

- The shared S-boxes are arranged in a linear array. These sharings must be correct and incomplete.
- Each share at the output of S-box i is made uniform by bitwise adding to it one or two shares from the input of S-box $i-1$.
- The state is augmented with d dummy components, called *guards*, to be added to the output of the first S-box in the array.
- The new value of the guards are taken from the input of the last S-box in the array.
- Uniformity is proven by giving an algorithm that computes the shared input from the shared output of this mapping.

For threshold sharings that have a so-called multi-permutation property, the guards can be reduced in size and so does the amount of bits fed forward.

1.2 Notation

Assume we have a nonlinear mapping that consists of a layer of invertible S-boxes. We denote the width of the S-boxes by n and their total number by m. So the layer operates on an array of $n \times m$ bits. We denote the input as $x = (x_1, x_2, x_3, \ldots x_m)$ and the output as $X = (X_1, X_2, X_3, \ldots X_m)$, with each of the x_i and X_i an n-bit array.

In general the S-boxes can differ per position. We denote the S-box at position i by S_i, so $X_i = S_i(x_i)$.

We denote addition in GF(2) by $+$.

1.3 Overview of the paper

In Section 2 we explain and prove the soundness of the method applied to the simplest possible case. In Section 3 we formulate the method for a more general case and in Section 4 we apply it to the nonlinear layer used in KECCAK, KEYAK and KETJE.

2 The basic method applied to 3-share threshold schemes

Assume the same S-box is used for all positions and its algebraic degree is 2 over GF(2), that we denote by S. In that case it is trivial to find a correct and incomplete threshold scheme with 3 shares S by substituting the terms in the algebraic expression of the S-box by their sum as components and appropriately distributing the monomials over the three shares of the S-box sharing. We denote the three shares that represent x_i by a_i, b_i and c_i, with $x_i = a_i + b_i + c_i$. Likewise, we denote the three shares that represent X_i by A_i, B_i and C_i, with $X_i = A_i + B_i + C_i$. The sharing of S consists of three functions from $2n$ to n bits, that we denote as (S_a, S_b, S_c). Correctness is satisfied if:

$$S_a(b,c) + S_b(a,c) + S_c(a,b) = S(a+b+c)\,.$$

Incompleteness is implied by the fact that each of the three elements of (S_a, S_b, S_c) take only two shares as inputs. In this scheme our m-component input x is represented by triplet (a,b,c) with three shares.

At the basis of our *Changing of the Guards* technique for achieving uniformity is the expansion of the shared representation. In particular, for the input we expand share b with an additional dummy component that we denote as b_0 and do the same for c. In this sharing x is represented by (a,b,c) where a has m components and both b and c have $m+1$ components. A triplet (a,b,c) is a uniform sharing of x if all possible values (a,b,c) compliant with x are equiprobable. As there are $2^{(3m+2)n}$ possible triplets (a,b,c) and being compliant with x requires the satisfaction of mn independent linear binary equations, there are exactly $2^{(3m+2)n-mn} = 2^{2(m+1)n}$ encodings (a,b,c) of any particular value x. The same holds for the sharing (A,B,C) of the output X.

Definition 1. *The* Changing the of Guards *sharing of an S-box layer where (S_a, S_b, S_c) is a sharing of S, mapping (a,b,c) to (A,B,C), is given by:*

$$\begin{aligned}
A_i &= S_a(b_i, c_i) + b_{i-1} + c_{i-1} && \text{for } i > 0\\
B_i &= S_b(a_i, c_i) + c_{i-1} && \text{for } i > 0\\
C_i &= S_c(a_i, b_i) + b_{i-1} && \text{for } i > 0\\
B_0 &= c_m\\
C_0 &= b_m
\end{aligned}$$

The sharing is depicted in Figure 1.

We can now prove the following theorem.

Theorem 1. *If S is an invertible S-box and (S_a, S_b, S_c) is a correct and incomplete sharing of S, the sharing of Definition 1 is a correct, incomplete and uniform sharing of an S-box layer with S as component.*

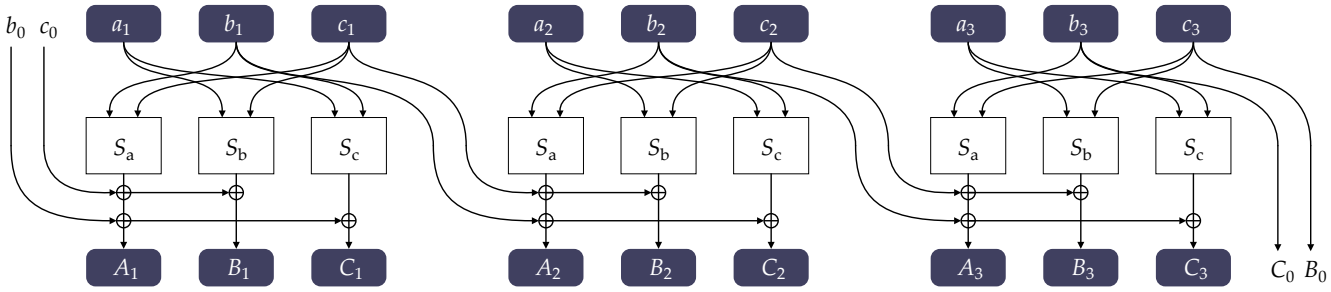

Fig. 1. Changing of the Guards sharing applied to simple S-box layer.

Proof. Correctness follows from the fact that each input components that is fed forward to the output of its neigboring components is added twice. We have for all $i > 0$:

$$\begin{aligned} A_i + B_i + C_i &= S_a(b_i, c_i) + b_{i-1} + c_{i-1} + S_b(a_i, c_i) + c_{i-1} + S_c(a_i, b_i) + b_{i-1} \\ &= S_a(b_i, c_i) + S_b(a_i, c_i) + S_c(a_i, b_i) \\ &= S(a_i + b_i + c_i)\,. \end{aligned}$$

For incompleteness, we see in Definition 1 that the computation of A_i does not involve components of a, the one of B_i does not involve components of b and the one of C_i does not involve components of c.

For uniformity, we observe that for each input x or each output X there are exactly $2^{2(m+1)n}$ valid sharings. If the mapping of Definition 1 is an invertible mapping from (a, b, c) to (A, B, C), it implies that if (a, b, c) is a uniform sharing of x, then (A, B, C) is a uniform sharing of X. It is therefore sufficient to show that the mapping of Definition 1 is invertible. We will do that by giving a method to compute (a, b, c) from (A, B, C).

We compute the components of (a, b, c) starting from index m down to 0. First we have $b_m = C_0$ and $c_m = B_0$. Then we can iterate the following loop for i going from m down to 1:

$$\begin{aligned} a_i &= S^{-1}(A_i + B_i + C_i) + b_i + c_i \\ b_{i-1} &= S_c(a_i, b_i) + C_i \\ c_{i-1} &= S_b(a_i, c_i) + B_i\,. \end{aligned}$$

□

The term "guards" refers to the dummy components b_0 and c_0 that are there to guard uniformity and that are "changed" to B_0 and C_0 by the shared implementation of the S-box layer.

The cost of this method is the addition of 4 XOR gates per bit of x and the expansion of the representation by $2n$ bits. The cost of additional XOR gates is typically not negligible but still relatively modest compared to the gates in the S-box sharing. For a typical S-box layer the expansion of the state is very small.

When applying this method to an iterated cipher that has a round function consisting of an S-box layer and a linear layer, one can do the following. The sharing of the S-box layer maps (a, b, c) to (A, B, C) and the linear layer is applied to the shares separately. In the linear mapping the guard components B_0 and C_0 are simply mapped to the components b_0 and c_0 of the next round by the identity.

3 Generalization to any invertible S-box layer

Here we give a method for an S-box layer with only restriction that the component S-boxes have the same width and are all invertible. We assume the S-boxes have maximum degree d and so we can produce a correct and incomplete threshold scheme with $d+1$ shares. We denote the shares by x^0 to x^d and component j of share i by x_i^j.

We now introduce d guard components: all but the first share get a guard component x_0^d.

Definition 2. *The* changing the guards *sharing of an S-box layer where $(S_i^0, S_i^1, \ldots, S_i^d)$ is a sharing of S_i, mapping $(x^0, x^1, x^2, \ldots, x^d)$ to $(X^0, X^1, X^2, \ldots, X^d)$, is given by (with $x_i \setminus x_i^j$ denoting the vector of d elements $x_i^{j'}$ for $j' \neq j$):*

$$
\begin{aligned}
X_i^0 &= S_i^0(x_i \setminus x_i^0) + x_{i-1}^{d-1} + x_{i-1}^d && \text{for } i > 0\\
X_i^1 &= S_i^1(x_i \setminus x_i^1) + x_{i-1}^d && \text{for } i > 0\\
X_i^2 &= S_i^2(x_i \setminus x_i^2) + x_{i-1}^1 && \text{for } i > 0\\
&\ldots\\
X_i^j &= S_i^j(x_i \setminus x_i^j) + x_{i-1}^{j-2} + x_{i-1}^{j-1} && \text{for } i > 0\\
&\ldots\\
X_i^d &= S_i^d(x_i \setminus x_i^d) + x_{i-1}^{d-2} + x_{i-1}^{d-1} && \text{for } i > 0\\
X_0^j &= x_m^{j+1} && \text{for } j > 0\\
X_0^d &= x_m^1
\end{aligned}
$$

We can now prove the following theorem.

Theorem 2. *If S_i are invertible S-boxes and $(S_i^0, S_i^1, \ldots S_i^d)$ are correct and incomplete sharing of S_i, the sharing of Definition 2 is a correct, incomplete and uniform sharing of the S-box layer with S_i as component.*

This theorem can be proven in a similar way as Theorem 1.

4 Application to the sharing χ' for Keccak

Keccak-p is the permutation underlying Keccak [11,2], Keyak [4] and Ketje [3] and is defined in [11,2]. Its nonlinear layer is called χ and has algebraic degree 2 over GF(2). In [1] a correct and incomplete sharing with 3 shares was proposed, denoted as χ'. The mapping χ operates independently on 5-bit rows and consequently χ' operates in parallel on 15-bit units. If we denote the elements of a row by s^0 to s^4 and the three shares by a, b and c: χ' is defined as (with concatenation denoting multiplication in GF(2), or equivalently, the binary AND operation):

$$A^x = b^x + (b^{x+1}+1)b^{x+2} + b^{x+1}c^{x+2} + b^{x+2}c^{x+1}, \tag{1}$$

$$B^x = c^x + (c^{x+1}+1)c^{x+2} + c^{x+1}a^{x+2} + c^{x+2}a^{x+1}, \tag{2}$$

$$C^x = a^x + (a^{x+1}+1)a^{x+2} + a^{x+1}b^{x+2} + a^{x+2}b^{x+1}, \tag{3}$$

with indexing taken modulo 5. Note that the state of Keccak-p is a three-dimensional array and we omit the y and z indices for clarity as we look here at a single row.

The mapping χ' has a remarkable property that we can exploit to reduce the overhead due to the "changing the guards" method. We call this a *multi-permutation property* inspired by [12].

We first need to introduce some notation. For a 5-bit vector s, let $\mathrm{L}(s) \triangleq (s^0, s^1, s^2)$ and and $\mathrm{R}(s) \triangleq (s^3, s^4)$. Similarly, we define $\mathrm{L}(a,b,c) \triangleq (a^0, b^0, c^0, a^1, b^1, c^1, a^2, b^2, c^2)$ and $\mathrm{R}(a,b,c) \triangleq (a^3, b^3, c^3, a^4, b^4, c^4)$.

Lemma 1. *For any choice of* $\mathrm{L}(A,B,C), \mathrm{R}(a,b,c)$ *there is exactly one solution* $\mathrm{L}(a,b,c), \mathrm{R}(A,B,C)$ *such that* $(A,B,C) = \chi'(a,b,c)$.

Proof. We describe how to compute $\mathrm{L}(a,b,c), \mathrm{R}(A,B,C)$ from $\mathrm{L}(A,B,C), \mathrm{R}(a,b,c)$. We rewrite each of the Equations (1) by switching lefthand term and first terms on the righthand from side:

$$\begin{aligned} a^x &= B^x + (b^{x+1}+1)b^{x+2} + b^{x+1}c^{x+2} + b^{x+2}c^{x+1}, \\ b^x &= C^x + (c^{x+1}+1)c^{x+2} + c^{x+1}a^{x+2} + c^{x+2}a^{x+1}, \\ c^x &= A^x + (a^{x+1}+1)a^{x+2} + a^{x+1}b^{x+2} + a^{x+2}b^{x+1}, \end{aligned}$$

We can use these equations for computing (a^2, b^2, c^2) by taking $i = 2$. Clearly the first term on the righthand side is part of $\mathrm{L}(A,B,C)$ and the remaining terms are expressed in terms of bits in $\mathrm{R}(a,b,c)$. We can now use this equation with $x = 1$ to compute (a^1, b^1, c^1) using the acquired value of (a^2, b^2, c^2). This can be repeated for $x = 0$ giving us the full knowledge of (a,b,c). From (a,b,c) we can compute (A,B,C) using Equations (1) and hence we also know $\mathrm{R}(A,B,C)$. □

We can use Lemma 1 to apply a variant of the *changing the guards* method to χ' that requires less state expansion and XOR gates due to the feedforward. We call it χ''.

Definition 3. *The* χ'' *sharing of* χ *is given by:*

$$\begin{aligned} \mathrm{R}(A_i) &= \mathrm{R}(\chi'_a(b_i, c_i)) + \mathrm{R}(b_{i-1}) + \mathrm{R}(c_{i-1}) && \text{for } i > 0 \\ \mathrm{R}(B_i) &= \mathrm{R}(\chi'_b(a_i, c_i)) + \mathrm{R}(c_{i-1}) && \text{for } i > 0 \\ \mathrm{R}(C_i) &= \mathrm{R}(\chi'_c(a_i, b_i)) + \mathrm{R}(b_{i-1}) && \text{for } i > 0 \\ \mathrm{L}(A_i) &= \mathrm{L}(\chi'_a(b_i, c_i)) && \text{for } i > 0 \\ \mathrm{L}(B_i) &= \mathrm{L}(\chi'_b(a_i, c_i)) && \text{for } i > 0 \\ \mathrm{L}(C_i) &= \mathrm{L}(\chi'_c(a_i, b_i)) && \text{for } i > 0 \\ \mathrm{R}(B_0) &= \mathrm{R}(c_m) \\ \mathrm{R}(C_0) &= \mathrm{R}(b_m)\,. \end{aligned}$$

Here the indexing i assumes the rows are arranged in a one-dimensional array. In Keccak-p *this is a two-dimensional array indexed by y and z. It is however simple to adopt a convention for converting this to a single-dimensional one, e.g. $i = y + 5z$.*

Note that $\mathrm{L}(b_0)$, $\mathrm{L}(c_0)$, $\mathrm{L}(B_0)$ and $\mathrm{L}(C_0)$ do not occur in the computations. We can therefore reduce the guards to their 2-bit right parts: $\mathrm{R}(b_0)$, $\mathrm{R}(c_0)$, $\mathrm{R}(B_0)$ and $\mathrm{R}(C_0)$.

The total expansion of the state reduces from 2 times the S-box width (totalling to 10 bits) to 4 bits. Moreover, there are only 8 XOR gates per S-box, i.e. 1.6 per native bit instead of 4 additional XOR gates per native bit. In the context of the χ' sharing the computational overhead is very small, as implementing Equation (1) requires 9 XOR gates and 9 (N)AND gates per native bit. Note that the multi-permutation technique can be applied to other primitives that use a variant of χ as nonlinear layer.

We can now prove the following theorem.

Theorem 3. *χ'' as defined in Definition 3 is a correct, incomplete and uniform sharing of χ.*

Proof. Correctness and incompleteness is immediate. For proving uniformity we describe how to compute (a, b, c) from (A, B, C). We compute the components of (a, b, c) starting from index m down to 0. First we have $\mathrm{R}(b_m) = \mathrm{R}(C_0)$ and $\mathrm{R}(c_m) = \mathrm{R}(B_0)$. Then we can iterate the following loop going from m down to 1, computing $a_i, \mathrm{L}(b_i), \mathrm{L}(c_i)$ and $\mathrm{R}(b_{i-1}), \mathrm{R}(c_{i-1})$:

- $\mathrm{R}(a_i) = \mathrm{R}(S^{-1}(A_i + B_i + C_i)) + \mathrm{R}(b_i) + \mathrm{R}(c_i)$
- compute $\mathrm{L}(a_i, b_i, c_i)$ from $\mathrm{L}(A_i, B_i, C_i), \mathrm{R}(a_i, b_i, c_i)$ using Lemma 1
- $\mathrm{R}(b_{i-1}) = \mathrm{R}(S_c(a_i, b_i)) + \mathrm{R}(C_i)$
- $\mathrm{R}(c_{i-1}) = \mathrm{R}(S_b(a_i, c_i)) + \mathrm{R}(B_i)$.

□

5 Conclusions

In this paper we introduce a simple and low-cost technique for achieving a $d + 1$-share correct, incomplete and uniform threshold implementation of any S-box layer of invertible S-boxes that have degree at most d. Looking for S-boxes with uniform threshold implementations with the minimum $(d + 1)$ number of shares has therefore lost much, if not all, of its relevance. However, it may become interesting now to look for S-boxes that have $d + 1$-share implementations with the multi-permutation property.

5.1 Acknowledgements

I thank Gilles Van Assche, Vincent Rijmen, Begül Bilgin, Svetla Nikova and Ventzi Nikov for working with me on the paper [5], that already contained an idea very close to the "changing of the guards" technique, Guido Bertoni for inspiring discussions and Lejla Batina for proofreading this text.

References

1. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *Building power analysis resistant implementations of* Keccak, Second SHA-3 candidate conference, August 2010.
2. ———, *The* Keccak *reference*, January 2011, `http://keccak.noekeon.org/`.
3. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer, *CAESAR submission:* Ketje *v2*, September 2016, `http://ketje.noekeon.org/`.
4. ———, *CAESAR submission:* Keyak *v2, document version 2.2*, September 2016, `http://keyak.noekeon.org/`.
5. Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche, *Efficient and first-order DPA resistant implementations of Keccak*, Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers (Aurélien Francillon and Pankaj Rohatgi, eds.), Lecture Notes in Computer Science, vol. 8419, Springer, 2013, pp. 187–199.
6. Joan Daemen, *Spectral characterization of iterating lossy mappings*, Cryptology ePrint Archive, Report 2016/090, 2016, `http://eprint.iacr.org/`, to appear in proceedings of SPACE 2016.
7. P. C. Kocher, J. Jaffe, and B. Jun, *Differential power analysis*, Advances in Cryptology – Crypto '99 (M. Wiener, ed.), Lecture Notes in Computer Science, vol. 1666, Springer, 1999, pp. 388–397.
8. S. Nikova, V. Rijmen, and M. Schläffer, *Secure hardware implementation of nonlinear functions in the presence of glitches*, ICISC (P. J. Lee and J. H. Cheon, eds.), Lecture Notes in Computer Science, vol. 5461, Springer, 2008, pp. 218–234.
9. ———, *Secure hardware implementation of nonlinear functions in the presence of glitches*, J. Cryptology **24** (2011), no. 2, 292–321.

10. Svetla Nikova, Christian Rechberger, and Vincent Rijmen, *Threshold implementations against side-channel attacks and glitches*, Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings (Peng Ning, Sihan Qing, and Ninghui Li, eds.), Lecture Notes in Computer Science, vol. 4307, Springer, 2006, pp. 529–545.
11. NIST, *Federal information processing standard 202, SHA-3 standard: Permutation-based hash and extendable-output functions*, August 2015, `http://dx.doi.org/10.6028/NIST.FIPS.202`.
12. Claus-Peter Schnorr and Serge Vaudenay, *Parallel FFT-hashing*, Fast Software Encryption, Cambridge Security Workshop, Cambridge, UK, December 9-11, 1993, Proceedings (Ross J. Anderson, ed.), Lecture Notes in Computer Science, vol. 809, Springer, 1993, pp. 149–156.