Provable Security Proofs and their Interpretation in the Real World

Vikram Singh [*]

Abstract

This paper analyses provable security proofs, using the EDL signature scheme as its case study, and interprets their benefits and drawbacks when applied to the real world.

Provable security has been an area of contention. Some, such as Koblitz and Menezes, give little credit to the potential extra security provided and argue that it is a distracting goal. However, others believe that an algorithm with a security proof is superior to one without it, and are prepared to accept the impact to performance that their use might involve. Goldreich has been notable for his defence of the security proof, and for his opposition to the view of Koblitz and Menezes. This paper is designed to help the reader make their own decisions on security proofs. We achieve this by giving an introduction to the typical security model used, then give a description of the EDL signature scheme and its tight reduction to the CDH problem in the Random Oracle Model, then analyse the proof's assumptions, meaning, validity and overhead for real world security.

Keywords: Cryptography, Provable Security, EDL Signature Scheme, Tight Reduction, Computational Diffie Hellman problem, Random Oracle Model

[*] `vs77814@gmail.com`. Completed in 2008 based on studies at RWTH Aachen University, 2005-2007.

Contents

1 Introduction

The field of Security Proofs is a daunting one to an outsider. The methodologies can seem strange, and the field can produce results that are incomprehensible to common sense, such as the extra bit used in the Katz Wang version of Schnorr's algorithm that provides twice the bit-length of security. Such results cause some to dismiss the field. However, proofs also give useful confidence in an algorithm's security, and many researchers regard this as being of utmost importance. As such, one finds strong positive and negative views on the matter.

Papers produced by Menezes and Koblitz ([3] and [4]) have attempted to critically examine aspects of security proofs and the claims of those that produced them. Unsurprisingly, these papers have met with fierce opposition, most notably in [8] from Goldreich who is vehement in his rebuttal of their ideas. However, behind the controversy is a rigorous mathematical discipline that is flawless in its own right. Whether it may be applied in the real world is where the debate probably truly lies.

This paper first introduces the subject, describing the security model which allows the difficulty of breaking a cryptographic algorithm to be related to the difficulty of solving an assumed-hard mathematical problem. Such a relation is called a reduction, and it is "tight" if the time to solve each is similar. A tight reduction thus enables security bounds to be put on the cryptographic algorithm.

The focus of this paper will then be the EDL signature algorithm, and consideration of the real world interpretation of the results of security proofs. EDL and the subsequent variants by Katz Wang and Chevallier-Mames have shown security proofs with the most success of late, with EDL being the first signature algorithm to have a tight reductionist security proof, thanks to Goh and Jarecki in [2]. This EDL proof will be described and analysed. We will not consider algorithms with asymptotic and non-tight security reductions as they provide comparatively insignificant benefit.

This paper is divided into six sections. Section 2 describes the principles behind producing security proofs. Section 3 describes the EDL signature algorithm and adaptations of the algorithm. In Sections 4 and 5 the security proof is described and analysed. The paper concludes in Section 6.

2 The Security Model

The purpose of a security proof is to show that within some *defined mathematical world*, if a machine is able to break the algorithm then that machine may be applied to solve a known hard problem (except with small probability). It is usually straightforward to show that breaking some hard problem (such as finding a discrete logarithm) results in breaking the algorithm, though not always. Thus, in the mathematical world we are reducing the problem of attacking the algorithm to solving a hard problem and may conclude that breaking the algorithm is at least an equally hard problem. This type of proof of security is also known as a reduction proof, and certainly the latter name is more accurate.

To produce such a proof, two protagonists are required; an *Adversary* and a *Simulator*. The role of the Adversary is to be able to produce a forgery in some way. The role of the Simulator is to use this ability of the Adversary to build a machine that solves the hard problem. The Simulator is named thus as it simulates the expected world of the Adversary so that the Adversary may not detect that it is being used for this purpose. The reasoning allows the logical conclusion that the Adversary will act in the same way to produce its forgeries whether inside the Simulator or not.

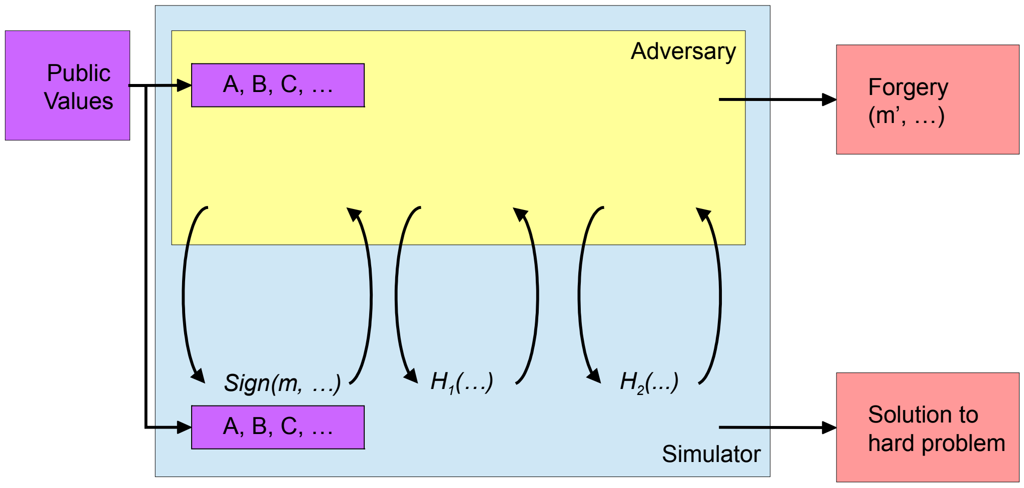

Figure 1: Adversary-Simulator Model

Figure 1 provides a pictorial explanation of the setup. Functions required by, but unknown to, the Adversary are simulated by the Simulator. The Simulator may choose the return values in any way so long as they are distributed as the Adversary expects. The next stage of these proofs is to produce a finite classification of all possible actions of the forger (A, B, C, etc. in the diagram). For each of these, it is demonstrated that the machine is either able to solve the hard problem or the forger is highly unlikely to produce a successful forgery. The prover then relates the probability/time to produce the forgery with that to solve the known hard problem.

2.1 The Random Oracle Model

In the security proof of most algorithms, it is assumed that the mathematical world is the *Random Oracle Model.* The Random Oracle Model assumes that the hash function may be chosen randomly by the Simulator from the set of all possible functions. In other words the hash is perfect, allowing the following properties:

1. The output of the hash function is indistinguishable from random output.
2. The Adversary's attack must be independent of the hash function used.
3. The Adversary cannot exploit any properties of the hash function.

The second of these points means that while the Adversary may use any public values, the Adversary may not know the hash functions used in the algorithm. The Adversary's knowledge of the hash functions would render any attempted simulation of the functions detectable. This point will be returned to in Section 5.

3 The EDL Algorithm

The design of EDL, as specified and proved in [2], is similar to Schnorr's algorithm, [7]. In fact one can view the algorithm as Schnorr's algorithm run in parallel on both the generator g and the message hash h. This is not accidental as the security proof of Schnorr's algorithm works by considering more than one iteration of the algorithm. By putting these iterations in parallel the designers of EDL are able to achieve tighter security bounds than may be achieved with Schnorr's algorithm, at the cost of a more complex algorithm.

3.1 Setup

The algorithm is performed in a cyclic group $\mathbb{G}$ of order q, generated by g. Let elements of $\mathbb{G}$ have bit length l_p and q be a prime with bit length l_q. Define l_r to be the bit-length of the random value r. The group is viewed multiplicatively. Given the space of messages is $\mathbb{M}$, two hash functions are required; $H : \mathbb{M} \times \{0,1\}^{l_r} \to \mathbb{G}$ and $G : \mathbb{G}^6 \to \mathbb{Z}_q$. Finally, it must be possible to generate random values of bit length l_q, and random values of bit length l_r.

3.2 Key Generation

The private key is a random number $x \in \mathbb{Z}_q$. The public key is $y = g^x$.

3.3 Signature Generation

To sign a message $m \in \mathbb{M}$, the signer generates $r \in \{0,1\}^{l_r}$, and $k \in \mathbb{Z}_q$. The signer computes $h = H(m,r)$ and $z = h^x$. Similarly, the signer computes $u = g^k$ and $v = h^k$. The signer then calculates the hash of these values $c = G(g,h,y,z,u,v)$ and computes $s = k + cx \mod q$. The signer produces the signature on m: $\sigma = (z,r,s,c)$.

3.4 Signature Verification

To verify the signature, $\sigma = (z,r,s,c)$, on a message $m \in \mathbb{M}$, the verifier computes $\tilde{h} = H(m,r)$, $\tilde{u} = g^s y^{-c}$ and $\tilde{v} = h^s z^{-c}$. The signature is accepted iff $c = G(g,\tilde{h},y,z,\tilde{u},\tilde{v})$.

3.5 Adaptations of the Algorithm

There are at least two further adaptations of this algorithm that the reader should be aware of. The first is the Katz Wang signature algorithm, [5]. This algorithm makes the improvement of reducing the size of r to a single(!) bit at the cost of a slightly weaker security bound. There is also the Chevallier-Mames signature algorithm proposed in [1]. This impressively alters the algorithm in a way that retains the security bounds while removing the need for r altogether. Both of these algorithms have a security proof that follow similar lines to the proof of EDL given in Section 4.

4 The Security Proof of the EDL Algorithm

The proof for the EDL algorithm precisely follows the model described in Section 2. The proof assumes the *Random Oracle Model* and that the Simulator and Adversary have the public values g, q, $\mathbb{G}$ and y. The proof then builds the Simulator and categorizes the possible actions of the Adversary. It then demonstrates that on production of a forgery, the Simulator is highly likely to be able to solve the following hard problem:

Definition 4.1. *The CDH problem* - The CDH or Computational Diffie Hellman problem is the problem of finding g^{ax} given g, g^a and g^x. Note that for EDL, g and $y = g^x$ are public values. It is assumed that $w = g^a$ is another public value known to the Simulator. Assume that in $\mathbb{G}$, given time $\tilde{\tau}$ the probability of solving the CDH problem is $\tilde{\epsilon}(\tilde{\tau})$.

4.1 The Simulated Functions

There are three functions that the Simulator answers for the forger, the two hashes $H(m, r)$, $G(g, h, y, z, u, v)$ and the signature simulation: $Sign(m)$.

- $H(m, r)$ - If the request has previously been made, the Simulator returns the previously returned value. Otherwise, the Simulator generates a random value $d \in \mathbb{Z}_q$ and returns: wg^d. Note that this is distributed randomly in $\mathbb{G}$ and hence H is indistinguishable from a random hash function.

- $G(g, h, y, z, u, v)$ - If the request has previously been made, the Simulator returns the previously returned value. Otherwise, the Simulator returns a random value: $\tilde{c} \in \mathbb{Z}_q$. Hence G is indistinguishable from a random hash function.

- $Sign(m)$ - If the request has previously been made, the Simulator returns the previous signature. Otherwise, the Simulator generates a random number $r \in \{0, 1\}^{l_r}$. If $H(m, r)$ has already been chosen the Simulator fails. Otherwise the Simulator generates a random number $\kappa \in \mathbb{Z}_q$ and sets $h = H(m, r) = g^\kappa$ and computes $z = y^\kappa (= h^x)$. The Simulator now chooses c and s at random and computes $u = g^s y^{-c}$ and $v = h^s z^{-c}$. If $G(g, h, y, z, u, v)$ has been previously chosen then the Simulator fails. Otherwise set $c = G(g, h, y, z, u, v)$ and return:

  $$\sigma = (z, r, s, c)$$

  Note that this signature is indistinguishable to the Adversary from one produced by a real signer.

Note that the Simulator may set the return values of G and H and thus produce signatures precisely because we are in the Random Oracle Model. The reader may concern themselves with the issue that the Simulator's control over the hash means it is able to produce forgeries itself, and so the reader may wonder why it needs the Adversary at all? The answer to this question is found by noting that the Simulator provides the adversary with a different output to H than it uses itself. The Adversary is in a more constrained position than the Simulator; the point being that success under these constraints is highly likely to result in a solution to the CDH problem.

4.2 Categorization of the Adversary's Actions

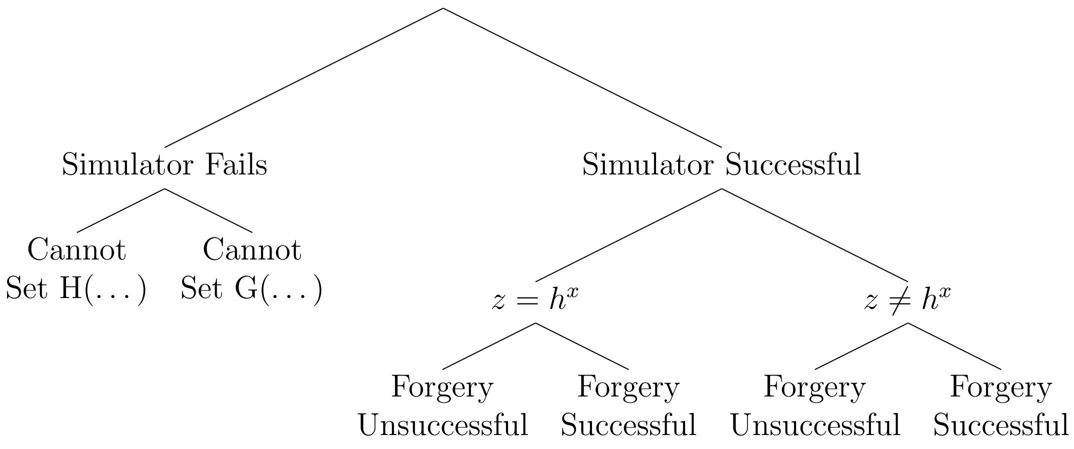

Figure 2: Adversary's actions

The Adversary's actions will be separated into six disjoint cases. These cases can be seen in Figure 2.

Firstly, in attempting to produce a signature, the Simulator may fail. If the Simulator fails, it is acting in a way not expected by the Adversary meaning the CDH will not be solved. There are two ways in which in could fail, due to $H(m, r)$ being previously requested or due to $G(g, h, y, z, u, v)$ being previously requested. Let q_H, q_G and q_S be the number of H, G and signature queries respectively, where hashes queried during signing do not count towards q_H or q_G. Then the probability of the first failure is at most $q_S \frac{q_H + q_S}{2^{l_r}}$.

If the second failure occurs then $G(g, h, y, z, u, v)$ has already been set. If this had been set as part of a previous signature query, a collision on H has occurred and the same $k = (s - cx)$ has been chosen by the Simulator. This occurs with probability at most $\frac{q_S^2}{q^2}$. Otherwise, this has been set by an Adversary's query to G. As the first failure did not occur, h must have been chosen at random. Furthermore, as the input must match with an input of the form; $(g, g^\kappa, y, y^\kappa, g^k, g^{k\kappa})$, and κ and k may be chosen randomly, the probability of this occurring is at most $\frac{q_S q_G}{q^2}$.

Thus the probability of Simulator failure is at most:

$$\mathbb{P}(\text{Sim Fails}) \leq q_S \left(\frac{q_H + q_S}{2^{l_r}} + \frac{q_G + q_S}{q^2} \right)$$

Now assume the Simulator is successful. If the Adversary produces a forgery on a message that is either incorrect or has previously been signed by the Simulator, then the Adversary has failed. If instead the Adversary has generated a successful forgery, $\sigma = (z, r, s, c)$, for a previously unsigned message, m, then the Adversary is successful. Assume that given time τ, the Adversary is successful with probability $\epsilon(\tau)$, so that:

$$\mathbb{P}(\text{Valid Forge}|\text{Sim Success}) = \epsilon(\tau)$$

There are two cases to consider: $z = h^x$ and $z \neq h^x$.

If $z \neq h^x$, then the Adversary must find c and s such that $c = G(g, h, y, z, g^s y^{-c}, h^s z^{-c})$. Assume $u = g^k$, $v = h^{\tilde{k}}$, $z = h^{\tilde{x}}$ for some $k, \tilde{k}, \tilde{x} \in \mathbb{Z}_q$ and $x \neq \tilde{x}$. As the forgery is valid, $c = G(g, h, y, z, u, v)$, $k = s - cx$ and $\tilde{k} = s - c\tilde{x}$. Thus there is only one valid c for each s as:

$$c = \frac{k - \tilde{k}}{\tilde{x} - x}$$

Thus the probability of the Adversary's success in this case is:

$$\mathbb{P}(\text{Valid Forge}|z \neq h^x \text{ \& Sim Success}) \leq \frac{q_G}{q}$$

If $z = h^x$, then the Simulator may solve the CDH problem. To produce the signature, the Adversary must have queried $H(m, r)$ or else there's only a $\frac{1}{q}$ probability of a valid forgery. Thus $h = wg^d$ for some d known to the Simulator. Hence the Simulator may calculate:

$$zy^{-d} = h^x g^{-dx} = w^x g^{dx} g^{-dx} = g^{ax}$$

If an exponentiation of a group element requires time τ_{exp}, then the runtime of the Simulator in this case is approximately: $\tau + \tau_{exp}(6q_S + q_H)$.

4.3 Reducing to a Hard Problem

Recall that in $\mathbb{G}$, given time $\tilde{\tau}$ the probability of solving the CDH problem is $\tilde{\epsilon}(\tilde{\tau})$. So set $\tilde{\tau} = \tau + \tau_{exp}(6q_S + q_H)$, and let F be the event that the forgery is valid, and S be the event that the Simulator is successful (with $\bar{S}$ being the complement), then:

$$\begin{array}{rcl}
\tilde{\epsilon}(\tilde{\tau}) & \geq & \mathbb{P}(z = h^x \;\&\; F \;\&\; S) \\
& = & \mathbb{P}(F \;\&\; S) - \mathbb{P}(z \neq h^x \;\&\; F \;\&\; S) \\
& = & (\mathbb{P}(F|S) - \mathbb{P}(z \neq h^x \;\&\; F|S))\mathbb{P}(S) \\
& \geq & (\mathbb{P}(F|S) - \mathbb{P}(F|z \neq h^x \;\&\; S))(1 - \mathbb{P}(\bar{S})) \\
& \geq & \mathbb{P}(F|S) - \mathbb{P}(F|z \neq h^x \;\&\; S) - \mathbb{P}(\bar{S}) \\
& \geq & \epsilon(\tau) - \dfrac{q_G}{q} - q_S\left(\dfrac{q_H + q_S}{2^{l_r}} + \dfrac{q_G + q_S}{q^2}\right)
\end{array}$$

Meaning that, except for a small error term, the probability the Adversary produces a forgery is less than the probability of solving the CDH problem in a similar period. By choosing parameters sensibly and by assuming knowledge of the most likely attack on the CDH problem, the equation above can be rearranged to give a non-trivial upper bound on the probability of success in time τ.

$$\epsilon(\tau) \leq \tilde{\epsilon}(\tilde{\tau}) + \frac{q_G}{q} + q_S\left(\frac{q_H + q_S}{2^{l_r}} + \frac{q_G + q_S}{q^2}\right)$$

One may like to believe that this is roughly an equivalence, but interestingly being able to solve the CDH problem does not imply that the Adversary can forge a signature. In this case, the Adversary can calculate the correct z given h and y, however the Adversary cannot publish s $(= k + cx)$ as x is unknown. Only g^s may be found.

We now note that for typical real-world bounds on the number of hash and signature queries available to the Adversary, the probability of CDH success is similar to the probability of forgery success, and the time taken in both cases is similar too. This gives the tight reduction of the forgery to a solution of CDH, which can be quantified once read-world bounds are assigned.

5 Analysis of the Security Proof

The proof has produced the impressive result of tightly relating the security of a signature algorithm to solving the CDH problem, given the assumptions (and the correctness of the proof). To interpret this result as providing security of the algorithm, it is necessary to attempt to embed the result into the real world.

5.1 Accuracy of Assumptions

At the beginning of the proof the Random Oracle Model was assumed. This attack restricts the Adversary to only finding attacks for a general hash function. However, as no algorithm is implemented using a general hash function, at some point it is clear that the designer will need to be specific. In this case the Random Oracle Model assumption is clearly incorrect unless it is proven that the best attack against the chosen hash function is generic. Until such a hash exists, it is difficult to perceive an entirely reliable application of these results.

Alternatively, one could take the view that the proof has ensured that only attacks that involve exploiting properties of the hash need be considered in assessing the security of the algorithm in the real world. *But does not considering attacks which apply with a generic hash make it easier to assess an algorithm?*

5.2 Attacking the Algorithm

Given that the Random Oracle Model is assumed, the simplest way to assess the value added by the proof is to consider attacks on the algorithm, and how these attacks are considered within the proof.

- H collision attack: Given a message, signature pair m, (z, r, s, c), find a $(\tilde{m}, \tilde{r})$ such that $H(m, r) = H(\tilde{m}, \tilde{r})$.
- G collision attack: Pick $z \in \mathbb{G}$ at random, find (s, c) such that $c = G(g, h, y, z, g^s y^{-c}, h^s z^{-c})$.
- G collision, H inversion attack. Find (m, r) such that $H(m, r) = g^k$, find (s, c) such that $c = G(g, g^k, y, y^k, g^s y^{-c}, (g^s y^{-c})^k)$ to achieve a forgery.
- Outside Group 'attack': Set $z = 0$, find (s, c) such that $c = G(g, h, y, 0, g^s y^{-c}, 0)$.

Careful consideration will show that all of these attacks except the last are considered by the Security Proof. Furthermore, sensible parameter checking would ensure that the last attack is not feasible.

The 'outside group attack' does raise an interesting point. Assuming (probably incorrectly) that the algorithm designer did not consider the case $z = 0$. The missing edge case occurs due to the probable implementation being over a ring rather than a group. Such subtle changes are out of scope of the security proof but have as much significance to the real security of the algorithm.

Alternatively, the security proof could be considered to have changed the problem from finding a flaw in the algorithm to finding a flaw in the proof of the algorithm or a hidden flaw in the assumptions (such as: $z \neq 0$). Thus the question remains: *Is it harder to find flaws in the algorithm's proof/assumptions than in the algorithm itself ?*

Furthermore, if it is, how much harder? For example, in the extreme, if there existed a signature algorithm which has had similar level of effort into breaking as the CDH problem, would a lack of reduction to the CDH problem be a concern as our assurance of the 'hardness' of the problems may be the same. If it were shown that breaking this new EDL algorithm is tightly reducible to breaking the long standing DSA, would this be of value?

5.3 Redundancy for Security

Comparing the 'G collision attack' and the 'G collision, H inversion attack' it appears that calculating the correct value for z achieves little for the attacker. Define $F(h, z, s, c) = G(g, h, y, z, g^s y^{-c}, h^s z^{-c})$. As G is a general hash function, we may assume that F is as well.

An attacker is attempting to find a (z, s, c) such that $c = F(h, z, s, c)$. While it is clear there is a solution for the correct z, this is also expected for any other x as F is a random function. Is there a purpose for z other than to be used as part of the security proof? It is worth the reader noting that removing x from EDL results in a randomised hashing version of Schnorr's signature algorithm.

6 Conclusion

Whatever your answer to the questions in Section 5, it is clear that a security proof cannot be perceived as a replacement for common sense analysis. A security proof is provided within a mathematical domain and hence careful consideration will always be required before its results can have significance in reality. In particular, the assumptions made as part of the Random Oracle Model cannot currently be satisfied and the significance of this will need consideration. Furthermore, efforts in attacking the algorithm could be perceived to have been mapped to attacking the proof or assumptions and this may or may not be of benefit. Lastly, the algorithm could contain redundancy whose presence has no perceivable security benefit other than to complete the proof.

References

[1] B. Chevallier-Mames. An Efficient CDH-based Signature Scheme With a Tight Security Reduction. 2005.

[2] E.-J. Goh and S. Jarecki. A Signature Scheme as Secure as the Diffie Hellman Problem. *Advances in Cryptology - EUROCRYPT 2003*, *Lecture Notes in Computer Science*, pages 401-415. Springer-Verlag, 2003.

[3] N. Koblitz and A. Menezes. Another look at "provable security". *Cryptology ePrint Archive*, Report 2004/152, 2004.

[4] N. Koblitz and A. Menezes. Another look at "provable security" II. *Cryptology ePrint Archive*, Report 2006/229, 2006.

[5] J. Katz, and N. Wang. Efficiency improvements for signature schemes with tight security reductions. *ACM Conference on Computer and Communications Security*, pages 155-164. ACM Press, 2003

[6] J. Stern, D. Pointcheval, J. Malone-Lee and N. Smart. Flaws in applying proof methodologies to signature schemes. *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 93-110. Springer-Verlag, 2002.

[7] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 1991.

[8] O. Goldreich. On Post-Modern Cryptography. 2006. *Cryptology ePrint Archive*, Report 2006/461, 2006.