ANOTHER LOOK AT AUTOMATED THEOREM-PROVING

NEAL KOBLITZ

ABSTRACT. I examine the use of automated theorem-proving for reductionist security arguments in cryptography and discuss three papers that purport to show the potential of computer-assisted proof-writing and proof-checking. I look at the proofs that the authors give to illustrate the "game-hopping" technique — for Full-Domain Hash signatures, El-Gamal encryption, and Cramer-Shoup encryption — and ask whether there is evidence that automated theorem-proving can contribute anything of value to the security analysis of cryptographic protocols.

Keywords: Automated theorem-proving, proof-checking, public key cryptography, signatures, encryption.

AMS classification: 03B35, 68T15, 94A60, 11T71

1. INTRODUCTION

The pre-program of Crypto '96 included an optional excursion to the beautiful Channel Islands off the coast near Santa Barbara. The boat trip featured close-up encounters with a myriad of sea creatures — several types of whales, seals, and porpoises, and fish and invertebrates of every size and description.

Unfortunately, the seas were rough. As the day wore on, most of the participants got badly seasick. There were just three hardy souls among the passengers — Josh Benaloh, Yvo Desmedt, and Ann Hibner Koblitz — who were unaffected by the rhythmic rocking of the boat.[1] Standing at the bow, they talked and laughed and enjoyed the cool spray as water splashed over them onto the deck. They were the only ones who appreciated seeing the varied sea creatures approaching the boat as more and more distressed passengers leaned over the side to "feed the fishes."

Just before the trip ended, the participants (those who were in a condition to write) were given a feedback questionnaire. Ann and Josh were near the crew when they read the responses. The crew doubled over in laughter as, one after another, under "Needs Improvement" they read comments like "seas need to be calmer," "ocean was too rough," "waves were too high." It sounded as if the passengers expected the boat operators to do something

[1] I went on the excursion, but my memories of the day can be summarized by the Beatles' line

> All the way the paper bag was on my knee,...

(Paul McCartney, "Back in the USSR," *The White Album*, 1968).

about that — as if there were some dial that needed to be adjusted or some computer software that needed to be debugged. To the crew, these responses fit in perfectly with their stereotype of the "technonerd" who believes that every inconvenience or discomfort in life must have a technological fix.

In this paper I examine another situation in which cryptographers naively hope that a suitably designed device — some cleverly written computer code — will make unpleasant natural phenomena go away. In this case the unpleasant phenomena are the tedium of writing out detailed proofs of reductionist security theorems and the embarrassment when those proofs are later found to have errors and gaps. The proposed technological fix is automated theorem-proving.

2. History

The story of attempts to automate theorem-proving goes back very far. According to Stillwell ([29], p. 469), "Since the time of Leibniz, and perhaps earlier, attempts have been made to mechanize mathematical reasoning." These efforts became more intense after the systematic development of symbolic logic by Boole and Frege in the 19th century. In this early period the culmination of the "formalist" program was the publication in 1910 of *Principia Mathematica* by Bertrand Russell and Alfred North Whitehead.

The main motive for the first wave of mathematical formalism was not, however, to make it possible for a machine to prove a theorem. Rather, the idea was to provide a guarantee of rigor that could be checked in a purely mechanistic way, without the need for intuition or experience. In fact, one would need to know only the list of axioms and the principles of logical deduction. In theory one would not have to know what objects, if any, were supposed to have the properties in the axioms and theorems. As Russell notoriously put it, "Mathematics is a science in which we never know what we are talking about nor whether what we say is true."[2]

In 1931 Gödel's proof of his Incompleteness Theorem took the wind out of the sails of the formalist philosophy of mathematics. But it did not refute the notion that a sufficiently detailed mathematical proof — such as the ones in *Principia Mathematica* — looks more like the result of a mechanical process than the product of high-level human thought.

In the computer age some people started to search for ways to mechanize certain types of proofs in a much more literal sense. In pure mathematics a watershed event was the announcement of the proof of the famous Four Color Conjecture at the American Mathematical Society summer meeting in 1976. For the first time in the proof of a major theorem, a computer was used to run through a large number of cases. Mathematicians did not react to this new application of computer technology with unbridled enthusiasm. As D. J. Albers reported in *The Two-Year College Mathematics Journal*,

[2]In our day this sentiment is most likely to be expressed by American students, but they don't mean it in the same sense as Bertrand Russell.

> Mathematician after mathematician expressed uneasiness with a proof in which a computer had played a major role. They were bothered by the fact that more than 1000 hours of computer time had been expended in checking some 100,000 cases and often suggested (hoped?) that there might be an error buried in the hundreds of pages of computer print outs. Beyond that concern was the hope that a much shorter proof could be found. Yu. I. Manin's observation [[23], p. 48] that "a proof only becomes a proof after the social act of 'accepting it as a proof'" fits the situation very well.

In the three decades since that time the role of computer-assisted proofs in pure mathematics has been extremely modest, except in certain unusual subspecialties that are far from the interests of most mathematicians. According to one promoter of automated theorem-proving [30], "The most exciting recent success in mathematics has been the settling of the Robbins problem by the ATP system EQP. In 1933 Herbert Robbins conjectured that a particular group of axioms form a basis for Boolean algebra, but neither he nor anyone else (until the solution by EQP) could prove this."[3]

In contrast, in mainstream mathematical research computer-assisted theorem-proving has been conspicuous by its absence, much to the surprise of many non-mathematicians. After Andrew Wiles' announcement of the proof of Fermat's Last Theorem, some reporters were disappointed to learn that computers had played no role whatsoever in the proof.

(Note: Here I am not referring to attempts to formally verify proofs that have already been written out in detail by mathematicians. For a recent example of this see *A formally verified proof of the prime number theorem* by Avigad, Donnelly, Gray, and Raff [1].)

Outside of mathematics, the main uses of automated theorem-proving, according to [30], are in "software generation and verification, protocol verification, and hardware verification." For example [30],

> NASA uses ATP to certify safety properties of aerospace software that has been automatically generated from high-level specifications. Their code generator produces safety obligations that are provable only if the code is safe. An ATP system discharges these obligations, and the output proofs, which can be verified by an independent proof checker, serve as certificates.

What is meant here (see also [12]) is a method of partially automating the verification that software will function as it's supposed to; it is a bit of a stretch to call that "theorem-proving." As in other areas of the so-called

[3]It is hard to resist the temptation to say sarcastically, "Be still, my beating heart!"

Artificial Intelligence field, extravagant terminology has outpaced actual accomplishments.[4]

* * *

The formalistic approach to security analysis of the protocols of public-key cryptography goes back almost as far as the notion of public-key cryptography itself. The first version of such a formal calculus is usually attributed to Dolev and Yao in 1983 [13].

The Dolev-Yao approach to analyzing public-key protocols is abstract and schematic. It views cryptographic primitives as idealized black boxes that must be assumed to have certain properties in an absolute sense. It is based on an all-or-nothing view of security and so has limited applicability in practical settings.

In recent years, especially since the publication of [3], researchers have been trying to bring theory closer to practice by giving reductionist security arguments that include precise inequalities involving probabilities and running times and that, in principle, can lead to concrete recommendations for key lengths and parameter selection.

The process of interpreting a reductionist security argument from the standpoint of cryptographic practice is fraught with difficulties and pitfalls, as Menezes and I argued in [19, 20]. Some of these problems come from fallacies or ambiguities in the security reductions themselves.

Among researchers in "provable security" the quality-control problem has been widely acknowledged. According to Halevi [16],

> The problem is that as a community, we generate more proofs than we carefully verify (and as a consequence some of our published proofs are incorrect).

Bellare and Rogaway [5] write:

> In our opinion, many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor.

Opinions differ as to the causes of this crisis. In [26] Nowak says that

> ...proofs [are] error-prone and difficult to check... because cryptographic schemes are based on non-trivial mathematics such as number theory, group theory or probability theory.

This explanation does not seem credible. After all, if this were the reason, then why is there no "crisis of rigor" in mathematics itself, a field that is also "based on non-trivial mathematics"? More plausible explanations are given by Bellare and Rogaway [5], who say that this crisis "has at its core a significant cultural element," and by Halevi [16], who acknowledges that

[4] In 1979 DeMillo, Lipton, and Perlis [11] published a thoughtful analysis of both technical and social issues involved in automated verification of computer programs. Their article is still relevant and interesting to read three decades later.

> Some of the reasons for this problem are social (e.g., we mostly publish in conferences rather than journals).

I also comment on the cultural and social issues in [17] and [18].

Whatever the reasons for the crisis in theoretical cryptography, the hope of authors of recent papers on automated theorem-proving is to take the uncertainties, flaws, and human foibles out of the process of deriving reductionist security arguments. These authors adopt a computational model that enables them to keep track of probabilities and running times as they go step-by-step from one "game" to another. The idea is to gradually transform the realistic scenario into an idealized one where the adversary has zero chance of succeeding. Often an error is introduced and recorded as one goes from one "game" to the next one ("game-hopping," as it is sometimes called). The sum of these errors can then be interpreted as the probability that the initial "game" differs from the last one — in other words, it is a bound on the probability that the adversary succeeds. This approach, which became popular after [5] and [27] appeared in 2004, is currently thought to be the direction of research that is most likely to produce automated proofs of security that have practical significance.

I shall examine three of these papers ([6], [26], and [16]) with particular attention to the examples those authors give to show the potential of their methods. One of these applications is to signatures, and the other two are to encryption.

3. Full-Domain Hash

The basic Full-Domain Hash (FDH) signature scheme [4] is one of the simplest and most elegant constructions in cryptography. Let $f : S \to S$ be a permutation of a finite set S. We assume f to be a *trapdoor one-way* function. In other words, using the public information any randomized algorithm has negligible probability (taken over the elements of S and the sets of coin tosses in executing the algorithm) of finding $f^{-1}(y)$ in a reasonable amount of time. Using secret information, f can easily be inverted.

Let h be a hash function — a function from message strings of arbitrary length to the set S — whose values range uniformly over the entire set S. (This is what "full-domain" means.) In addition, we suppose that the hash function is indistinguishable from a random function from message strings to S. That is, the hash function can be modeled by an "oracle" that returns random values in S whenever it is queried for a value $h(M)$. The only condition on the oracle is that if the same M is queried again, it must return the same value $h(M)$.

The basic signature scheme works as follows. The signer Alice possesses a secret key that allows her to invert f. To sign a message M, she finds $h(M)$ and then $s = f^{-1}(h(M))$, which is her signature. After receiving M and s, Bob verifies the signature by checking that $f(s) = h(M)$. That's all there is to it.

Now suppose that an adversary that does not have Alice's secret key wants to forge a signature. Because the values $h(M)$ are assumed to be random and uniformly distributed over S, the adversary cannot do this with non-negligible probability unless it can (with non-negligible probability) invert f, and this is precisely what the one-wayness assumption rules out. Thus, signatures are unforgeable in this sense if the assumptions (one-wayness of f and randomness of h) hold.

What one wants, however, is a stronger notion of unforgeability. Namely, we suppose that the adversary is allowed to ask for a bounded number (no more than q_s) of signatures of messages of its choice. It is considered to be a successful adversary — an "existential forger under chosen-message attack" — if it produces a valid signature of some message of its choice other than the ones whose signatures it has been given.

As Menezes and I argued in [19], from a common-sense standpoint it is still clear that signatures are unforgeable in this strong sense provided that the one-wayness assumption for f and the randomness assumption for h hold. The reason is that the freedom that the adversary has to query signatures of messages of its choice and then forge any message of its choice is illusory, because it has no control over the hash values of those messages. The chosen-message queries result in a set of f^{-1}-values of random elements h_i of S, and then it is required to find f^{-1} of some unrelated random element $y \in S$. This is no easier than finding $f^{-1}(y)$ without knowing f^{-1} of those other random values. To see this, simply note that both sets $\{h_i\}$ and $\{s_i\}$, where $s_i = f^{-1}(h_i)$, are distributed uniformly at random in S. The set of pairs (h_i, s_i) could equally well have been generated starting with the set $\{s_i\}$, that is, as $(f(s_i), s_i)$. The adversary could have done this without making any queries. Thus, the chosen-message attack gives it nothing useful.

However, many people are not satisfied with an intuitive argument such as this one; they want a reductionist security argument. That is, they want to know that if an existential forger under chosen-message attack exists, then that adversary can be used as a subroutine to construct an algorithm that violates the underlying assumption that f is a one-way permutation. This was proved by Bellare and Rogaway in [4].

Here is how the reduction in [4] works. Suppose that we are given Alice's public key and an arbitrary $y \in S$, and we are asked to find x such that $y = f(x)$. We must show how we could find x (with high probability) if we had a forger that can mount chosen-message attacks.

So suppose that we have such a forger. We give it Alice's public key and wait for its queries. It asks for the hash values of at most q_h messages and for signatures of at most q_s of these messages. In all cases but one, we respond to the hash query for a message M_i by randomly selecting $x_i \in S$ and setting the hash value h_i equal to $f(x_i)$. For just one value M_{i_0} we respond to the hash query by setting $h_{i_0} = y$. We choose i_0 at random and hope that M_{i_0} happens to be the message whose signature will be forged by our existential forger. Any time the adversary makes a signature query for

a message M_i with $i \neq i_0$ we send x_i as its signature. Notice that this will satisfy the forger, since $f(x_i) = h_i$. If the forger ends up outputting a valid signature s_{i_0} for M_{i_0}, that means that we have the solution $x = s_{i_0}$ to our original equation $y = f(x)$.

If we guessed wrong and M_{i_0} is not the message that the forger ends up signing, then we won't be able to give a valid response to a signature query for M_{i_0} if the forger requests that signature. The forger either will fail or will give us useless output, and we have to start over again. We also have to start over again if the forger succeeds in forging a signature for a message M_i with $i \neq i_0$, since that just gives us information we already know, namely, $x_i = f^{-1}(h_i)$. Recall that q_h is the bound on the number of queries of the hash function. In any iteration of the procedure we have a probability of success at least $1/q_h$. If we go through k iterations, the probability that every single time we fail to solve $y = f(x)$ for x is at most $(1 - 1/q_h)^k$. For large k — namely, for k greater than $O(q_h)$ — this approaches zero; so with high probability we succeed. This completes the argument.

The original paper [4] dealt with RSA-FDH, where the one-way permutation f is given by $f(x) = x^e \bmod N$. In that case Coron [8] showed how to reduce the number of iterations needed from $O(q_h)$ to $O(q_s)$, where q_s is the bound on the number of signature queries in the chosen-message attack. Here is his idea. Suppose that when the adversary makes its q_h hash queries, we choose a greater number of values h_i — not just the single value h_{i_0} — in a way that will later enable us to find the e-th root of $y \bmod N$ if M_i turns out to be the forged message. Namely, for some randomly chosen subset I_0 consisting of $[q_h/u]$ indices we choose a corresponding set of random values z_i, and then we respond to a hash query for M_i with $i \in I_0$ by setting $h_i = z_i^e y$. For $i \notin I_0$, as before we choose x_i at random and set $h_i = x_i^e \bmod N$. Note that if the adversary ends up forging a signature for one of the $[q_h/u]$ messages M_i with $i \in I_0$, then we need only divide the signature s_i by z_i to get the e-th root of $y \bmod N$.

What should we choose u to be? We have to be careful not to make it too small. Each time the adversary makes a signature query for a message M_i, there is approximately a $1/u$ chance that $i \in I_0$, in which case h_i is one of the $[q_h/u]$ hash values for which we do not know an e-th root $\bmod N$. In that case, we cannot answer the query, and we must restart the adversary. The chance that we will not have to do this for any of the q_s signature queries is roughly $(1 - \frac{1}{u})^{q_s} \approx e^{-q_s/u}$. This means that we should not choose u much smaller than q_s. Note that the probability of success of a single iteration is $\frac{1}{u}(1 - \frac{1}{u})^{q_s}$; if we choose $u \approx q_s$, then each iteration has a probability of success roughly equal to $1/q_s$.

From a practical standpoint it makes a world of difference whether the forger algorithm must be repeated q_h times or q_s times. Because signature queries require a response from the target of the attack, one can realistically suppose that their number is somewhat limited — say, $q_s \approx 2^{20}$. In contrast,

a hash query corresponds in practice to evaluating a publicly available function. There is no justification for assuming that an attacker's queries to the hash oracle are bounded by anything smaller than the running time. To be safe one should take $q_h \approx 2^{80}$. Thus, the Bellare-Rogaway result for RSA-FDH implies that we are guaranteed 80 bits of security only if we choose our parameters so that the RSA function f cannot be inverted in time less than 2^{160}. On the other hand, Coron's result implies that we are guaranteed that level of security provided that it is not possible in time less than 2^{100} to invert the RSA function. (Of course, if we don't insist on reductionist arguments and find the common-sense argument convincing, then we can be confident that FDH has 80 bits of security provided that f cannot be inverted in time less than 2^{80}.)

When q_s is very small, Coron gives us a tight reduction; in other words, he shows that in that case we get 80 bits of security for RSA-FDH if we choose parameters so that e-th roots mod$N$ cannot be found in time less than 2^{80}. In the special case of a passive attack — that is, $q_s = 0$ — Coron's argument is particularly simple. We respond to all of the attacker's hash queries by choosing z_i at random and setting $h_i = z_i^e y \bmod N$. No matter what message M_i it forges a signature for, we immediately get the e-th root of $y \bmod N$ by dividing the signature by z_i.

The RSA problem — that is, finding e-th roots mod$N$ — has a special property that makes Coron's argument possible: for fixed key (e, N) it is self-reducible. This means that any instance y can be "randomized" — it can be shifted to a random instance $z^e y$ such that a solution to the new instance will immediately enable us to solve the original instance. This property implies that all instances (except for a negligibly small subset) are equally hard.

The self-reducibility (for fixed key) of RSA-FDH is a nice property to have because it allows us to rule out the possibility that there is a small but non-negligible proportion ϵ of the set S on which the trapdoor one-way permutation $f : S \to S$ is relatively easy to invert and that this subset of S is easily identifiable. To see what the danger would be in the case of an f that is not self-reducible, let us suppose that for a proportion $\epsilon = 2^{-40}$ of values $y \in S$ it is possible to compute $f^{-1}(y)$ in 2^{40} steps. We are also supposing that the "easy" values of y can be quickly recognized. Then a passive attacker can run through roughly 2^{40} messages M, any one of which it would be happy to forge a signature for, until it gets a hash value $y = h(M)$ that is easy to invert. It then forges a signature in time 2^{40}.

Thus, it is not surprising that the two most important FDH-type signature schemes — RSA-FDH and the Boneh-Lynn-Shacham scheme [7] — both have the self-reducibility property for fixed key. It would be risky indeed to use an FDH scheme for which the difficulty of inverting f might be dramatically different for different values of $y \in S$.

Note also that in the situation described above a user Alice might easily fool herself into thinking that she has 80 bits of security. Suppose, for

instance, that the algorithm A to invert f takes a small amount of time T and that one iteration of A has probability $p(y)$ of finding $f^{-1}(y)$, where for fixed y the probability is taken over the sets of coin tosses in executing A. If $p(y) \approx 2^{-40}$ for a proportion $\epsilon = 2^{-40}$ of $y \in S$ and $p(y)$ is negligible for other y, then the probability of success of A taken over both instances y and sets of coin tosses is 2^{-80}. So Alice might believe that she has 80 bits of security, whereas in reality her signatures are existentially forgeable. From a mathematical standpoint Alice's fallacy is using multiplication rather than addition — $2^{40} \cdot 2^{40} = 2^{80}$ rather than $2^{40} + 2^{40} = 2^{41}$ — when she estimates her adversary's running time in the scenario described above. The best way to protect Alice from such a calamity is to use a trapdoor one-way permutation f that has the self-reducibility property.

* * *

In the paper [6] the main example that Blanchet and Pointcheval use to illustrate their approach to automated security proofs is the general Full-Domain Hash signature scheme. To the casual reader the authors give the impression that their automatic theorem-prover is essentially reconstructing the reductionist security argument of Bellare and Rogaway [4]. Indeed, in §1 they say that "the only way to 'formally' prove [security] is by showing that an attacker against the cryptographic protocol can be used as a subpart in an algorithm (the reduction) that can break the basic computational assumption"; and in §4.1 they describe the result of their theorem-prover as "the usual upper-bound" that's in [4]. However, a careful reader will discover that what the automated theorem-prover does in [6] is in fact much less than what Bellare and Rogaway did in [4].

In the first place, the automated theorem-prover in [6] does not give a reduction at all. It does not describe how to use an adversary as a subroutine in order to invert the one-way permutation. That is, the theorem-prover gives no indication of how a simulator should respond to the adversary's hash and signature queries. Instead, the theorem-prover uses essentially the same "common-sense" argument that I gave (see above) to justify the claim that the q_s signature queries couldn't possibly help the attacker. Namely, it notes that (in our notation) the sequences $(h, f^{-1}(h))$ and $(f(s), s)$ have the same distribution, i.e., they are "observationally equivalent." In other words, without loss of generality one can take $q_s = 0$. Once one is dealing only with a passive adversary, the rest of the argument boils down to the observation that, informally speaking, if p is the probability of inverting f in one iteration of an algorithm with running time T, then a forger that tries to find f^{-1} of no more than q_h random values has at most $q_h p$ probability of success in time roughly T.[5]

[5]This bound $q_h p$ can be reached in a situation where the adversary can quickly determine whether a random value is one of the $p|S|$ values for which f^{-1} can be found in time T; in that case "time roughly T" means time of the form $T + q_h t$ with t small.

In the second place, even though the factor q_h in the adversary's success probability is the same as in [4], it arises for a completely different reason. In [4] q_h is a "nontightness" factor coming from the reduction argument, whereas in [6] there is no reduction argument, and the factor q_h comes because the adversary has a choice of q_h hash values of which it need only find f^{-1} of any one. If (for a fixed key) f is equally hard to invert for all $y \in S$ (except for a negligibly small subset), then in time $\approx T$ the adversary has probability of success only p, not $q_h p$, and it takes the same amount of time to sign either a fixed message or any one of many messages. That is, it makes no difference whether each new iteration uses a new message or just a new set of coin tosses with the same message. If, on the other hand, there's a small but non-negligible subset of S on which f is relatively easy to invert and if elements of this subset are easy to recognize, then the adversary's success probability may be as great as $q_h p$. Thus, the appearance of the annoying factor q_h in [6] is caused by a totally different consideration than in [4], namely, the possibility of a poorly constructed f for which the inversion problem is not uniformly difficult.

In the third place, the automated proof in [6] cannot readily be modified to get a much better bound for RSA-FDH and other self-reducible versions of FDH. This is in sharp contrast to the Bellare-Rogaway proof for RSA-FDH, which can be modified in a natural way (see [8] and the discussion above) so as to give much tighter bounds — including a completely tight bound in the case of a passive attack.

In the fourth place, the security model in [6] uses a measure of probability that must be interpreted with great caution. Namely, the success probability of inverting $f : S \to S$ is assessed not only over the sets of coin tosses and the instances $y \in S$, but also over the keys that are generated from a random seed by the key-generator algorithm.

Let me describe two scenarios that show the possible pitfalls that arise from the use of this probability space. In both cases I suppose that we are in the following hypothetical situation. I assume that the only reasonable way to invert $f : S \to S$ is by recovering the private key, and that the fastest way to recover the private key is to run many iterations of a certain algorithm A. For reasonable parameter choices the algorithm A takes a small amount of time T, so that the running time to invert f is essentially equal to the number of iterations of A, and this is equal to the reciprocal of the probability that a single iteration of A will recover the private key. I further suppose that the key-generation algorithm usually produces a key pair that requires τ iterations to break, but a certain small proportion ϵ of the time it produces a key pair that can be broken in only $\rho \ll \tau$ iterations. Finally, I suppose that it is relatively easy to determine from the public key whether or not a key pair is one of the weak ones; that is, one has an algorithm to do this directly, rather than having to simply try out A with different keys.

For our first scenario suppose that $\tau = 2^{80}$, $\epsilon = 2^{-40}$, and ρ is small. Then the probability of inverting f in time ρT is of order 2^{-40}, so we might conclude that we have only 40 bits of security, which is worthless. However, if there are, say, about 2^{25} users, then an attacker would be over 99.99% certain *not* to be able to find a user with a weak key, in which case the implementation would in practice have 80 bits of security.

In contrast, for our second scenario suppose that $\tau = 2^{81}$, $\epsilon = 2^{-25}$, and $\rho = 2^{56}$. In this case the probability of inverting f in time T is about 2^{-80}, so we might be confident that we have 80 bits of security. However, since the weak keys can be readily identified, if there are over 2^{25} users an attacker has a good chance of finding a user with a weak key pair, after which it can obtain that user's secret key in 2^{56} operations.

Of course, since I am assuming that weak keys can be recognized directly, the key-generation algorithm could be modified so as to identify and not assign weak keys. However, in practice protocol designers who believe that they have a proof of security sometimes decide to increase efficiency by dispensing with key validation procedures that are not required in their proof. This was one of the central issues in the controversy over HMQV (see [21, 24, 25]). Moreover, an attacker might be willing to invest more resources in weak-key spotting than the designers of the system would want to put into the key-generation algorithm, since the latter would be under competitive pressure to reduce implementation time while an adversary would be much less limited. So there could be keys that are not weak enough to be spotted by the key-generation algorithm but are identified by the adversary as being much weaker than an average key.

In [20] Menezes and I commented that whether or not one is bothered by the possible existence of a small proportion of weak keys depends on one's point of view — for example, whether one is an individual user or a system administrator:

> ...let's suppose that in a certain cryptosystem a small proportion — say, 10^{-5} — of the randomly assigned private keys are vulnerable to a certain attack. From the standpoint of an individual user, the system is secure: she is 99.999% sure that her secret is safe. However, from the standpoint of the system administrator, who is answerable to a million users, the system is insecure because an attacker is almost certain...to eventually obtain the private key of one or more of the users, who will then sue the administrator.

Thus, in certain circumstances the probability of a forger's success as defined in [6] can be a misleading measure of the true security of the signature scheme. It might either lead Alice to doubt the security of a protocol that in practice is secure or, even worse, lead her to have confidence in the security of a protocol that is actually vulnerable to attack.

Finally, it is amusing to note that the theorem-prover used in [6] required 14800 lines of Ocaml code, and the summary by Blanchet and Pointcheval of what the theorem-prover did in the case of Full Domain Hash (Appendix C in their paper) occupies five single-spaced pages! The result of all this effort is a weak tautological statement that raises more questions than it answers.

What lessons can be learned from the above analysis of the security proof for FDH by the automated theorem-prover in [6]? The Blanchet-Pointcheval paper was accepted for presentation at Crypto, which is the most prestigious of the annual cryptography conferences, with an acceptance rate of only about 15%. The Program Committee apparently felt that this work represented one of the most impressive advances so far in the area of automated theorem-proving. Yet we have seen that the human-made reductionist security proof of Bellare and Rogaway, even though it is a rather crude early result, is much more satisfying from both practical and theoretical standpoints than the pale imitation that the automated theorem-prover in [6] was able to come up with. In the metaphorical race between human and machine, it seems that we human theorem-provers needn't worry about the automated ones snapping at our heels any time soon.

4. ElGamal encryption

In [26], Nowak describes "a framework in which proofs are precise enough to be mechanically checked, and readable enough to be humanly checked." He illustrates his approach to "game-based security proofs" by proving semantic security of ElGamal encryption under the Decision Diffie–Hellman assumption (DDH). Nowak explains the benefits of computer-assisted theorem-proving and proof-checking as follows:

> The main advantages of using a proof assistant are that reasoning errors are not possible and that all the assumptions must be stated. On the other hand all tedious details of the proof must be dealt with: you cannot simply claim that something is obvious.

In this section I discuss Nowak's proof of semantic security of ElGamal encryption (for brevity I will look only at the non-hashed variant) and examine both the significance of what is proved and the extent to which his objectives — human checkability and machine checkability — have been achieved.

The ElGamal encryption scheme, as described in [26], is as follows. Let G be a finite cyclic group of order q, let $g \in G$ be a generator, and let $\mathbb{Z}_q$ denote the integers modulo q. The scheme consists in three probabilistic algorithms: key generation, which chooses x randomly from $\mathbb{Z}_q$ and sets the public and secret keys (kp, ks) equal to (g^x, x); encryption, which, given kp and a message $m \in G$, chooses y randomly from $\mathbb{Z}_q$ and sets the ciphertext

c equal to the pair $(g^y, kp^y m)$; and decryption, which, given ks and a ciphertext c, computes $\pi_2(c)(\pi_1(c)^{ks})^{-1}$ (where π_1 and π_2 are the projections of the ciphertext onto its first and second components).

Before discussing Nowak's proof, let's recall in informal terms what semantic security means (in the sense of [15]) and how one can be convinced that ElGamal encryption is semantically secure assuming DDH. Semantic security (in an equivalent formulation known as indistinguishability under chosen plaintext attack, IND-CPA) assuming DDH means the following: if an adversary chooses two messages m_0 and m_1 and sends them to the **encrypt** algorithm, which encrypts one of them m_b chosen at random, then the adversary will not be able to guess b with probability significantly greater than $\frac{1}{2}$ — that is, with an "advantage" greater than some specified ϵ — unless it is able to solve DDH with probability greater than $\frac{1}{2} + \epsilon_{\mathrm{DDH}}$.

By definition, solving DDH in a group G with generator g means that, given a triple (g^x, g^y, g^z), one must determine whether or not $z \equiv xy \bmod q$, that is, whether or not $g^z = g^{xy}$. The DDH assumption says, roughly speaking, that there is no efficient algorithm that can do this with non-negligible probability, where the probability is taken over all instances of DDH and all random sequences of coin tosses used in the algorithm. In other words, there is no such algorithm whose probability of returning "true" when presented with input of the form (g^x, g^y, g^{xy}) differs by more than ϵ_{DDH} from its probability of returning "true" when presented with input of the form (g^x, g^y, g^z) with z random. If one has a Diffie-Hellman triple and a non-Diffie-Hellman triple and randomly chooses one of them as input to such an algorithm, it will correctly determine whether or not the triple is of the form (g^x, g^y, g^{xy}) with probability greater than $\frac{1}{2} + \epsilon_{\mathrm{DDH}}$.

It's intuitively clear that the adversary's task in trying to violate semantic security — namely, given G, g, g^x, and $c = (g^y, g^{xy} m_b)$ determine whether the last component gives a Diffie-Hellman triple after it is divided by m_0 or after it is divided by m_1 — is very similar to the problem of distinguishing between a Diffie–Hellman triple and a random triple. If one wants a formal argument, one can construct a reduction from DDH to breaking ElGamal encryption in the sense of semantic security. Namely, suppose that one has an adversary that violates semantic security with advantage $> \epsilon_{\mathrm{DDH}}$. Given a triple (g^x, g^y, α) where α may or may not be equal to g^{xy}, one sends the public key g^x to the adversary, which selects two messages m_0 and m_1. One then randomly chooses $b \in \{0, 1\}$ and sends the adversary the ciphertext $(g^y, \alpha m_b)$. If the adversary correctly determines b, then one outputs "true" (i.e., one's triple is a Diffie–Hellman triple); otherwise — that is, if it outputs the wrong b or else fails to give meaningful output, as could happen if $\alpha \neq g^{xy}$ and c is not an encryption of either message — one outputs "false." It is easy to see that the probability of outputting "true" is $\leq \frac{1}{2}$ when $\alpha \neq g^{xy}$ and $> \frac{1}{2} + \epsilon_{\mathrm{DDH}}$ when $\alpha = g^{xy}$.

The "proof of security" in the last paragraph will probably strike the reader as rather obvious. In fact, a reader who's in an uncharitable mood might even characterize it as a trivial tautology.

Now let's look at Nowak's proof. He starts out with "game-zero," which is the following formalized version of the definition of semantic security in the IND-CPA formulation. Here A_1 denotes the algorithm that the adversary uses to produce two messages m_0, m_1, and A_2 denotes the adversary's algorithm that guesses which bit is the subscript of the encrypted message; both A_1 and A_2 are probabilistic, and so depend on a random string r taken from some set R.

$$\left| \Pr_{=\text{true}} \left(\begin{array}{l} \text{let } (kp, ks) \Leftarrow \textbf{keygen}() \text{ in} \\ \text{let } r \leftarrow_R R \text{ in} \\ \text{let } (m_0, m_1) \leftarrow A_1(r, kp) \text{ in} \\ \text{let } b \leftarrow_R \{1, 2\} \text{ in} \\ \text{let } c \Leftarrow \textbf{encrypt}(kp, m_b) \text{ in} \\ \text{let } \hat{b} \leftarrow A_2(r, kp, c) \text{ in} \\ [\hat{b} =^? b] \end{array} \right) - \frac{1}{2} \right| \leq \epsilon_{\text{DDH}}$$

The structure of the proof is a series of "hops" from one game to an essentially equivalent one, until finally obtaining a game whose truth is an immediately consequence of one of Nowak's corollaries. (The earlier pages of [26] contain a library of fourteen theorems, propositions, and corollaries later to be used to effect the transition steps in the security proofs for ElGamal and hashed-ElGamal encryption.)

The transition from game-zero to game-one involves replacing **keygen** and **encrypt** by their definitions:

$$\left| \Pr_{=\text{true}} \left(\begin{array}{l} \text{let } x \leftarrow_R \mathbb{Z}_q \text{ in} \\ \text{let } (kp, ks) \leftarrow (g^x, x) \text{ in} \\ \text{let } r \leftarrow_R R \text{ in} \\ \text{let } (m_0, m_1) \leftarrow A_1(r, kp) \text{ in} \\ \text{let } b \leftarrow_R \{1, 2\} \text{ in} \\ \text{let } y \leftarrow_R \mathbb{Z}_q \text{ in} \\ \text{let } c \leftarrow (g^y, kp^y m_b) \text{ in} \\ \text{let } \hat{b} \leftarrow A_2(r, kp, c) \text{ in} \\ [\hat{b} =^? b] \end{array} \right) - \frac{1}{2} \right| \leq \epsilon_{\text{DDH}}$$

After three more games (which I will omit so as not to unduly tax the readers' time and try their patience), Nowak ends with a final game-five:

$$\Pr_{=\text{true}} \left(\begin{array}{l} \text{let } x \leftarrow_R \mathbb{Z}_q \text{ in} \\ \text{let } y \leftarrow_R \mathbb{Z}_q \text{ in} \\ \text{let } r \leftarrow_R R \text{ in} \\ \text{let } b \leftarrow_R \{1, 2\} \text{ in} \\ \text{let } z \leftarrow_R \mathbb{Z}_q \text{ in} \\ [A_2(r, g^x, (g^y, g^z \pi_b(A_1(r, g^x)))) =^? b] \end{array} \right) = \frac{1}{2}$$

This concludes the proof, because game-five is true according to a corollary that says that the probability of choosing a particular element $a \in A$ when an element is randomly chosen from A is equal to $1/|A|$.

Let's compare Nowak's proof of several pages with the one-paragraph one I gave before. His proof has the advantage of being machine-checkable using the French "Coq" proof-assistant. Probably Nowak is also correct in claiming that his proof is "humanly checkable," although that might depend on who the hapless human is who is assigned this tedious task.

However, in a deeper sense Nowak's six-game proof is much less suitable for human consumption than the one-paragraph proof that I wrote in plain English. The danger to humans reading his proof is that they will not see the forest for the trees. That is, they'll get so bogged down in verifying each "hop" from one game to another that they won't see the big picture. In this case the big picture is that what is being proved is little more than a trivial tautology. In any event, there can be no doubt that the game-hopping proof in [26] would have to be rated a 0 on a 0-to-10 scale if it is judged according to the criterion given at the end of the discussion of proofs in Yu. I. Manin's book on mathematical logic ([23], p. 51): "a good proof is one which makes us wiser."

* * *

Both my one-paragraph proof of the tautology and Nowak's six-game proof fail to make use of an important property of ElGamal encryption that implies a much stronger result. Namely, the discrete-logarithm and Diffie–Hellman problems in a fixed prime-order group have a self-reducibility property that allows us to make a specific instance into a random instance. As a result the reductionist argument can be strengthened to show that an adversary that guesses b with probability $> \frac{1}{2} + \epsilon$ can be used to construct a DDH-solver that succeeds not just with probability $> \frac{1}{2} + \epsilon_{\mathrm{DDH}}$, but in fact with probability $> 1 - \epsilon$.

To see this, suppose that we are given a triple (g^x, g^y, α) and want to determine whether or not $\alpha = g^{xy}$. For randomly selected i, j we send the adversary the public key g^{xi}, and we encrypt the message m_b (for a random $b \in \{0, 1\}$) as $(g^{yj}, \alpha^{ij} m_b)$. Note that $(g^{xi}, g^{yj}, \alpha^{ij})$ is a Diffie–Hellman triple if and only if (g^x, g^y, α) is one. After a large number of different pairs (i, j) we look at the proportion of correct guesses of b by the adversary. If our triple (g^x, g^y, α) is a Diffie–Hellman triple, then it is almost certain that the proportion of correct answers will be significantly greater than $\frac{1}{2}$; if not, then it is almost certain that the adversary will either not give useful output or else output the wrong guess close to 50% of the time. (I've omitted some details.) We thus have a method to solve DDH with almost 100% chance of success.

Unfortunately, this stronger and more interesting result does not seem to be readily obtainable or checkable using Nowak's techniques. The obstacle appears to be that the argument requires some thought and is not just a

sequence of trivial equivalences that can be looked up in a library of generic tautologies.

Finally, the reader might have noticed that my definition of ElGamal encryption at the beginning of this section — which I took directly from [26] — is missing a crucial condition, namely that the group order q is prime. This condition does not, of course, affect Nowak's proof of security, which is formal and generic. However, it vitally affects the real-world security of ElGamal. If q is smooth (that is, not divisible by a large prime number), then discrete logs in G are easy to find. If q has a large prime factor but also a small factor — for example, if G is the multiplicative group $\mathbb{F}^\times$ of a finite field $\mathbb{F}$ of characteristic $\neq 2$ — then the DDH assumption can easily be shown to be false, and ElGamal is not semantically secure.

It is quite common for authors of provable security papers to inadvertently neglect some important condition and thereby imperil the real-world security of the system. The presence of small subgroups in G or in a group that naturally contains G (for example, $\mathbb{F}^\times$ or the full group of points on an elliptic curve) can make a system vulnerable to attack. Just as Nowak in [26] apparently forgot about the need to choose prime-order G, similarly it was the omission of a public-key validation step that would have prevented small-subgroup attacks that led to flaws in some of Krawczyk's HMQV protocols (see [24, 25]). And in like manner the Bangerter–Camenisch–Maurer [2] zero-knowledge proof of possession of a discrete logarithm in a group of unknown order was shown in [22] to be flawed because nothing would prevent a verifier from using invalid parameters to break the protocol. These cases illustrate the danger of putting so much energy into ever more formalistic security proofs: while locking the front door tighter and tighter, people might easily fail to notice that the back door has swung wide open.

5. Cramer–Shoup encryption

In [16] Halevi illustrates the hoped-for benefits of computer-aided proofs of security by giving a detailed outline of how that would work for the classic reductionist security result of Cramer and Shoup [9, 10]. In this section I compare a non-computer-aided version of the proof (following the exposition in [19]) with Halevi's treatment. For brevity I shall discuss not the entire proof, but only one of the key steps in it.

We start by recalling the main result in [9, 10].

Reductionist security claim. If the Decision Diffie–Hellman Problem is hard in the group G and if the hash function H is collision-resistant, then the encryption scheme described below is indistinguishability-secure against chosen-ciphertext attack.

Description of the Cramer–Shoup encryption scheme. Let $x = (x_1, x_2)$, $y = (y_1, y_2)$, $z = (z_1, z_2)$ denote pairs of integers between 1 and $q - 1$; let $g = (g_1, g_2)$ and $u = (u_1, u_2)$ denote pairs of elements of G; and let r denote

a random integer between 1 and $q-1$. We use the notation $g^x = g_1^{x_1} g_2^{x_2}$, $g^{rx} = g_1^{rx_1} g_2^{rx_2}$, and so on.

The group G of prime order q in the field of p elements (where $q|p-1$) and two random non-identity elements g_1, g_2 are publicly known. We suppose that the collision-resistant hash function H takes triples of integers mod p to integers between 0 and $q-1$ (that is, it takes bit-strings of length $3\lceil \log_2 p \rceil$ to bit-strings of length $\lceil \log_2 q \rceil$). Alice's private key consists of three randomly generated pairs x, y, z, and her public key consists of the three group elements $c = g^x$, $d = g^y$, $e = g^z$.

To send a message $m \in G$, Bob chooses a random r, sets $u_1 = g_1^r$, $u_2 = g_2^r$, and $w = e^r m$, and computes the hash value $h = H(u_1, u_2, w)$ of the concatenation of these three mod p integers. He computes $v = c^r d^{rh}$, and sends Alice the ciphertext 4-tuple (u_1, u_2, w, v). In this ciphertext the element v allows Alice to check that Bob enciphered the message properly; w contains the message m "disguised" by the factor e^r; and u_1 and u_2 are the information Alice needs to remove this factor using her private key.

More precisely, to decipher the 4-tuple (u_1, u_2, w, v) Alice first computes $h = H(u_1, u_2, w)$ and uses her secret key to find u^{x+hy} (this is shorthand for $u_1^{x_1+hy_1} u_2^{x_2+hy_2}$), which should be equal to v (because $u^{x+hy} = g^{rx+ryh} = c^r d^{rh}$). If it is not equal to v, Alice rejects the message. If the ciphertext passes this test, she proceeds to decrypt by dividing w by u^z. Since $u^z = g^{rz} = e^r$ and $w = e^r m$, this gives her the plaintext m. This concludes the description of the cryptosystem.

We say that a 4-tuple (u_1, u_2, w, v) is invalid — not a possible ciphertext — if $\log_{g_1} u_1 \neq \log_{g_2} u_2$. Another way to say this is that the 4-tuple (u_1, u_2, w, v) is invalid if and only if the Decision Diffie–Hellman Problem has a negative answer for the 4-tuple (g_1, g_2, u_1, u_2). (Here we are using a different variant of DDH than in §4, where the generator was fixed and the input was a variable triple; now we let the generator g_1 vary, and so the input is a 4-tuple.)

One of the basic parts of the proof of the security claim is to show that an invalid ciphertext will almost certainly be rejected by Alice. We first give the argument of Cramer–Shoup [9, 10] in the form of a mathematical proof using logic and natural language. After that I discuss Halevi's approach using computer code.

In the sequel most discrete logarithms will be expressed to the base g_1. Define $\beta = \log_{g_1} g_2$. Then the discrete logs to the base g_1 of the elements c and d of Alice's public key are given by

$$(1) \qquad \gamma = \log_{g_1} c = x_1 + \beta x_2 \bmod q; \qquad \delta = \log_{g_1} d = y_1 + \beta y_2 \bmod q.$$

Let us regard the two pairs x, y as elements of a 4-dimensional vector space V over the field of q elements. Alice's public key constrains x, y to the 2-dimensional affine subspace S defined by the above two equations. Note that the values γ and δ are fixed once her public key is specified, although these values are not necessarily known even to Alice. For given values of the

public key and the public parameters, Alice's private 4-tuple (x_1, x_2, y_1, y_2) is equally likely to be any point of S.

If $\log_{g_1} u_1 = r$ and $\log_{g_1} u_2 = r\beta \bmod q$ for some r, then the ciphertext is valid. In that case the verification equation $v = u^{x+hy}$ holds either everywhere on S (if v has the value for which $\log_{g_1} v = r(\gamma + h\delta) \bmod q$), or else nowhere on S (if v has any other value). On the other hand, suppose that the ciphertext is invalid, that is, $\log_{g_1} u_1 = r$ and $\log_{g_1} u_2 = r'\beta \bmod q$ with $r' = \log_{g_2} u_2 \neq r$. Then the verification equation $v = u^{x+hy}$ holds if and only if

$$\log_{g_1} v = r(\gamma + h\delta) + \beta(r' - r)(x_2 + hy_2). \tag{2}$$

Regardless of the value of v, this equation imposes another independent linear condition on x, y, and so Alice accepts the 4-tuple (u_1, u_2, w, v) only if x, y lie on a particular line in S. The chance of that is $1/q$, which is negligible. Thus, Alice will almost certainly reject an invalid ciphertext.

In Halevi's version, the part about non-acceptance of invalid ciphertext comes at the end of the proof after the code has been modified by applying five different "games." Among many other things, these games have introduced the variables β, γ, and δ.[6] He now modifies the code by introducing a new variable X (denoted r' in [16]) for the right side of equation (2). He then modifies the code by inserting lines that treat the new variable X, along with the old variable y_2, as independently chosen and that define the variables x_2, x_1, y_1 in terms of the other variables. For example, one line of the new code reads as follows (in our notation);

$$\text{009b} \qquad x_2 \longleftarrow \frac{X - r(\gamma + h\delta)}{\beta(r' - r)} - hy_2 \bmod q$$

Halevi observes that $v = g_1^X$ now depends not on the earlier variables, but only on a newly-defined one, and this enables the definition of v to be moved to the end of the code where a "bad-flag" (corresponding to the "bad" event of accepting an invalid ciphertext) is located. After some more changes of variables and moving of code, Halevi has the code defining the "bad" event where he wants it.

In all this moving around of code and replacing one section of code by another one with changed variables, it is easy to lose sight of what stands out as the central point in the mathematical argument: There is negligible probability $1/q$ of accepting an invalid ciphertext because that's a codimension-1 event; that is, it happens only if Alice's secret key lies in a subspace of one dimension lower than that of the full space of her possible secret keys.

What argument does the code-based game in [16] use to justify this crucial probability step? None. Halevi simply decrees that "the tool can be endowed with a rule that lets it assign a probability of $1/q$ for this test..."

Thus, in order to be convinced of the logical correctness of the proof by computer game-playing, one has to already know the logic of the original

[6]These variables are denoted w, x, y in [16]; in what follows I have changed the notation to be consistent with [19].

proof. One also needs to understand the original proof in order to get the changes of variables right when writing the code. It is questionable whether this kind of code-based argument really represents progress toward automated theorem-proving or even proof-checking. The code could not have been constructed if the theorem had not already been proven, and correctness of the proof cannot be checked unless one already understands a key mathematical step (the codimension-1 argument). It is not even clear that the roughly ten pages of text and pseudocode in [16] can somehow help readers convince themselves of the correctness of the mathematical proof that was written in natural language. Rather, the reverse is much more likely — that is, it is only after thoroughly understanding and being convinced of the validity of the original proof that readers have any chance of being able to make sense of Halevi's code-based version.

Halevi's pseudocode for the different games he uses to establish the Cramer–Shoup result goes on for significantly more pages than the original proof in [9] or the later version in [10]. It is hard to see what has been achieved, other than to convert a moderately difficult proof into one that is very difficult to follow.

6. Conclusion

I do not wish to discourage anyone from taking a code-based approach to writing up reductionist security proofs or from conceptualizing such proofs as a series of hops from one game to another. It would be arrogant in the extreme for me to pass judgment on anyone's preferred style of proving things. That is a matter of personal taste. As Menezes and I commented in [19], cryptography is as much an art as a science. As in the arts, people must be free to experiment with new ways of expressing themselves. And in cryptography just as in the arts, fashionable trends from time to time veer toward the abstruse and arcane. The popularity of code-based proofs can be seen as an example of this.

One of the great insights of post-modernism [28] is that there is no such thing as scientific progress, and one should not speak of one approach being better or worse than a different one. Science, like other areas of human endeavor, is subjective and culturally determined, according to those who view the history of science as a more or less arbitrary sequence of shifts from one paradigm to another. Thus, if we want to adhere to the tenets of post-modern cryptography (a term that Oded Goldreich [14] obligingly introduced into common use), we should embrace *all* possible approaches to "proving" security of cryptographic protocols and should resist the temptation to call game-hopping and automated theorem-proving a step backward.

* * *

The problem with such terms as "automated theorem-proving," "computer-aided theorem-proving," and "automated proof-checking" is the

same as the problem with the term "provable security" (see [19]). In both cases they promise a lot more than they deliver.

Will we ever see a machine-checkable proof that would not have been easily checkable by humans if it had been written out clearly and carefully with proper attention to lucid expository style? Will we ever see a reductionist security argument devised by computer that was not obtained before by hand or that could not have been more easily derived without the computer? Until this happens, researchers in this area should expect to encounter a certain amount of healthy skepticism. So far the papers in this field leave the reader asking, in the words of an often-quoted advertisement for Wendy's fast-food restaurants, "Where's the beef?"

7. Acknowledgments

I would like to thank Ann Hibner Koblitz and Alfred Menezes for valuable comments on earlier drafts of this paper.

References

[1] J. Avigad, K. Donnelly, D. Gray, and P. Raff, *A formally verified proof of the prime number theorem*, to appear in the ACM Transactions on Computational Logic.

[2] E. Bangerter, J. Camenisch, and U. Maurer, *Efficient proofs of knowledge of discrete logarithms and representations in groups with hidden order*, Public Key Cryptography – PKC 2005, LNCS 3386, 2005, pp. 154-171.

[3] M. Bellare, *Practice-oriented provable security*, Proceedings of the First International Workshop on Information Security (ISW '97), LNCS 1396, Springer-Verlag, 1998, pp. 221-231.

[4] M. Bellare and P. Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*, Proceedings of the First Annual Conference on Computer and Communications Security, ACM, 1993, pp. 62-73.

[5] M. Bellare and P. Rogaway, *Code-based game-playing proofs and the security of triple encryption*, available at http://eprint.iacr.org/2004/331

[6] B. Blanchet and D. Pointcheval, *Automated security proofs with sequences of games*, Advances in Cryptology – Crypto 2006, LNCS 4117, Springer-Verlag, 2006, pp. 537-554.

[7] D. Boneh, B. Lynn, and H. Shacham, *Short signatures from the Weil pairing*, Advances in Cryptology – Asiacrypt 2001, LNCS 2348, Springer-Verlag, 2001, pp. 514-532.

[8] J.-S. Coron, *On the exact security of full domain hash*, Advances in Cryptology – Crypto 2000, LNCS 1880, Springer-Verlag, 2000, pp. 229-235.

[9] R. Cramer and V. Shoup, *A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack*, Advances in Cryptology – Crypto '98, LNCS 1462, Springer-Verlag, 1998, pp. 13-25.

[10] R. Cramer and V. Shoup, *Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack*, SIAM Journal of Computing 33 (2003), pp. 167-226.

[11] R. A. DeMillo, R. J. Lipton, and A. J. Perlis, *Social processes and proofs of theorems and programs*, Communications of the ACM 22 (1979), pp. 271-280.

[12] E. Denney, B. Fischer and J. Schumann, *Using automated theorem provers to certify auto-generated aerospace software*, Proceedings of Second International Joint Conference on Automated Reasoning, Lecture Notes in Artificial Intelligence 3097 (2004), pp. 198-212.

[13] D. Dolev and A. C. Yao, *On the security of public-key protocols*, IEEE Transactions on Information Theory, IT-29(2) (1983), pp. 198-208.

[14] O. Goldreich, *On post-modern cryptography*, available at http://eprint.iacr.org/2006/461.

[15] S. Goldwasser and S. Micali, *Probabilistic encryption*, Journal of Computer and System Sciences 28 (1984), pp. 270-299.

[16] S. Halevi, *A plausible approach to computer-aided cryptographic proofs*, available at http://eprint.iacr.org/2005/181.

[17] N. Koblitz, *The uneasy relationship between mathematics and cryptography*, Notices of the Amer. Math. Society 54 (2007), pp. 972-979, available at: http://www.ams.org/notices/200708.

[18] N. Koblitz, *Random Curves: Journeys of a Mathematician*, Springer-Verlag, 2007.

[19] N. Koblitz and A. Menezes, *Another look at "provable security,"* Journal of Cryptology 20 (2007), pp. 3-37; available at http://eprint.iacr.org/2004/152.

[20] N. Koblitz and A. Menezes, *Another look at "provable security." II*, Advances in Cryptology – Indocrypt 2006, LNCS 4329, Springer-Verlag, 2006, pp. 148-175; available at http://eprint.iacr.org/2006/229.

[21] H. Krawczyk, *HMQV: A high-performance secure Diffie-Hellman protocol*, Advances in Cryptology – Crypto 2005, LNCS 3621, 2005, pp. 546-566.

[22] S. Kunz-Jacques, G. Martinet, G. Poupard and J. Stern, *Cryptanalysis of an efficient proof of knowledge of discrete logarithm*, Public Key Cryptography – PKC 2006, LNCS 3958, 2006, pp. 27-43.

[23] Yu. I. Manin, *A Course in Mathematical Logic*, translated by N. Koblitz, Springer-Verlag, 1977.

[24] A. Menezes, *Another look at HMQV*, Journal of Mathematical Cryptology 1 (2007), pp. 47-64; available at http://eprint.iacr.org/2005/205.

[25] A. Menezes and B. Ustaoglu, *On the importance of public-key validation in the MQV and HMQV key agreement protocols*, Progress in Cryptology – Indocrypt 2006, LNCS 4329, 2006, pp. 133-147.

[26] D. Nowak, *A framework for game-based security proofs*, available at http://eprint.iacr.org/2007/199.

[27] V. Shoup, *Sequences of games: a tool for taming complexity in security proofs*, available at http://eprint.iacr.org/2004/332.

[28] A. D. Sokal, *Transgressing the boundaries: Toward a transformative hermeneutics of quantum gravity*, Social Text, 46/47 (Spring/Summer 1996), pp. 217-252.

[29] J. Stillwell, *Mathematics and Its History*, 2nd ed., Springer-Verlag, 2002.

[30] G. Sutcliffe, *An overview of automated theorem proving*, downloaded 23 March 2006 from http://www.cs.miami.edu/~tptp/OverviewOfATP.html

Department of Mathematics, Box 354350, University of Washington, Seattle, WA 98195 U.S.A.

E-mail address: koblitz@math.washington.edu