Seminar on Trustworthy Computing

November 3, 2004

DOES PROVABLE SECURITY EXIST?

Neal Koblitz, Department of Mathematics
koblitz@math.washington.edu

This talk is based on joint work with Alfred Menezes of the Centre for Applied Cryptographic Research at Waterloo; see
http://eprint.iacr.org/2004/152.pdf
or
http://www.cacr.math.uwaterloo.ca

Suppose that someone is using public-key cryptography to

- **protect credit card numbers in e-commerce;**
- **maintain confidentiality of medical records;**
- **safeguard national security information.**

How can she be sure that the system is secure?

Is there such a thing as a "proof" of security?

In 1994 Bellare and Rogaway "proved security" for a version of RSA that they called "Optimal Asymmetric Encryption Padding" (OAEP).

Soon after, MasterCard and Visa included OAEP in their SET electronic payment standard.

In 2001, Victor Shoup examined the Bellare-Rogaway "proof of security" and showed that it was fallacious.

This type of thing causes a credibility problem.

What might "provable security" mean?

Let's start with the word "security" (in encryption).

First answer:

Every public-key system is based on a one-way function or construction.

Example:

In RSA, this is multiplying together two large primes p and q to form the "modulus" n:

$$n = pq.$$

The inverse process is factoring n, which is presumably impractical.

So a first attempt at a definition of "security" is:

being certain that the underlying one-way function cannot feasibly be inverted.

However, from the earliest days of public-key cryptography (the late 1970s) it was realized that the RSA encryption function

$$y = x^e \pmod{n}$$

(where e is a fixed exponent, x is the plaintext and y is the ciphertext) could conceivably be inverted even if the underlying one-way function

$$(p, q) \mapsto n = pq$$

cannot be, that is, even if n is not factored.

(No one has any idea how to do this.)

Thus, a second answer to what "security" should mean is:

being certain that the encryption function cannot feasibly be inverted.

Most mathematically oriented books on cryptography don't go any further than these two answers.

For example, my two books on cryptography have this defect.

This limited view fails to anticipate most of the attacks on a cryptosystem that are likely to occur.

Example (Bleichenbacher 1998):

Suppose that Alice is receiving messages via RSA.

The plaintext messages must adhere to a certain format. If a deciphered message is not in that form, Alice's computer transmits an error message to the sender.

This seems innocuous.

But Bleichenbacher can (sometimes) use these error messages to decipher an RSA ciphertext WITHOUT knowing the deciphering key.

Outline of Bleichenbacher's attack:

Let y be the target ciphertext.

He sends Alice some carefully selected 'perturbations' y' of y and keeps a record of which ones are rejected because the decryption is not in the proper format.

He is able to put this information together to find the exact decryption of y.

Bleichenbacher's attack is an example of a "chosen-ciphertext" attack.

A full chosen-ciphertext attack means that the adversary is able to get Alice to decrypt any messages of his choice other than the target ciphertext y.

Bleichenbacher mounts a partial chosen-ciphertext attack, in the sense that he gets Alice to give him partial information about the decryption of any message of his choice (namely, whether or not it has the proper format).

In the 1980s — long before Bleichenbacher's attack — Goldwasser, Micali, Rivest and others realized that "security" should include the ability to withstand a chosen-ciphertext attack.

This is a much stronger notion of security, because the adversary is assumed to have a lot of power — namely, the ability to get a lot of information out of Alice.

What does "proof" of security mean?

The standard answer:

A reduction argument.

The idea of "reducing one problem to another" is common throughout mathematics and computer science.

Math example:

Ribet's Theorem (1985). Fermat's Last Theorem reduces to the Taniyama Conjecture.

That is, the Taniyama Conjecture is at least as hard as FLT.

Computer science example:

The theory of NP-completeness

To show that a problem $\mathcal{P}$ is NP-complete, it suffices to show that 3SAT reduces to it — in other words, that any algorithm to solve $\mathcal{P}$ can be used (with little additional effort) to solve 3SAT.

That implies that $\mathcal{P}$ is as hard as 3SAT — which means that it's NP-complete.

In cryptography, 3SAT is replaced by a mathematical problem that everyone believes to be hard — for example, factoring large integers; and $\mathcal{P}$ is the problem of breaking a given cryptosystem.

"Provable security" results have the conditional form:

If problem X is intractable, then the cryptosystem Y is secure against attacks of type Z.

Notice that

• the intractability of the underlying mathematical problem is being assumed (it's not being proved); and

• there is no assurance that cryptosystem Y will not succumb to an attack of type Z', where $Z' \neq Z$.

First example of a "provable security" reduction:

Rabin encryption.

Recall the RSA encryption function

$$y = x^e \pmod{n},$$

where e is a fixed exponent that is prime to $\varphi(n) = (p-1)(q-1)$ (where the modulus n is the product of the two very large primes p and q).

The message recipient Alice knows the factorization of n and thus can easily determine a decryption exponent d such that

$$x = y^d \pmod{n}.$$

However, it's conceivable that someone could find the plaintext x without knowing the factorization of n.

Rabin encryption (invented soon after RSA) is a variant of RSA where $e = 2$.

Since squaring is a 4-to-1 map on the integers mod n, Rabin has to modify things a little.

First, he specifies that both p and q be primes of the form $4k + 3$.

Second, he finds an easily identifiable set $\mathcal{S}$ consisting of 1/4 of the residues mod n, such that every square mod n has exactly one squareroot in $\mathcal{S}$.

A plaintext x must be an element of $\mathcal{S}$.

Alice decrypts by finding one of the squareroots of y modulo n and then adjusting it if it's not the right one, that is, if it is not in $\mathcal{S}$.

She can do this by multiplying by -1 (mod n) or by $\pm\varepsilon$ (mod n), where ε is given by

$$\varepsilon \equiv 1 \pmod{p},$$

$$\varepsilon \equiv -1 \pmod{q}.$$

Since Alice knows the factorization of n, she has no trouble computing $x \in \mathcal{S}$ such that $y \equiv x^2$ (mod n).

Unlike RSA, Rabin encryption can be shown to have the converse property:

Only someone who can factor n could perform Rabin decryptions.

Unlike RSA, Rabin encryption has the feature that breaking it is *provably* equivalent to factoring n, in the following sense:

THEOREM. Factoring n reduces to the problem of finding the plaintext from the ciphertext in Rabin encryption.

COROLLARY. If factoring is hard, then the Rabin cryptosystem is secure against inverting the encryption function.

PROOF OF THEOREM. Suppose that there's a computer program that, given the Rabin ciphertext y, finds the Rabin plaintext x.

You want to use this computer program to factor n.

Choose a random x_0 and set $y = x_0^2 \pmod{n}$.

Apply the computer program to y; it then gives you $x \in \mathcal{S}$ such that $x^2 \equiv y \equiv x_0^2 \pmod{n}$.

There's a 50% chance that $x \neq \pm x_0 \pmod{n}$, in which case you can compute $x_0/x = \pm\varepsilon$ and then immediately factor n:

$$p = \text{g.c.d.}(n, \varepsilon - 1).$$

Otherwise, try again with a new x_0.

After k different random x_0's, there's only a 2^{-k} chance that you've failed to factor n.

This is a probabilistic algorithm to factor n, given a program to get plaintext from ciphertext. QED

No such reduction is known for RSA.

In fact, in 1998 Boneh and Venkatesan showed that it is highly unlikely that there is a reduction from factoring to inverting the RSA encryption function.

More precisely, they showed that if e is small and if there's an "algebraic" reduction from factoring to inverting RSA, then the factoring problem itself must be easy.

Thus, Rabin encryption has a fundamental "provable security" property that RSA lacks.

Does that mean that Rabin encryption is "provably secure"?

Not exactly.

The same reasoning used to prove the above theorem also shows that Rabin encryption is totally vulnerable to chosen-ciphertext attack.

If an adversary can get Alice to decrypt $y = x_0^2 \pmod{n}$ for k different x_0 that he chooses at random, then with probability $1 - 2^{-k}$ he'll be able to factor n.

Depending on what definition of "security" we use, the same argument can "prove" either security... or insecurity!

NOTE: In 2001, Boneh showed how to modify Rabin's construction — with a "padding" and a "Feistel round" — in order to get an encryption system that is both efficient and arguably quite secure.

Now let's pass from encryption to signature.

Crucial ingredient: a "hash" function $H(m)$.

In general, a hash function is a function from a long string m (in cryptography, m is the message) to a much shorter string $H(m)$ that behaves like a "fingerprint" of the message.

H is a publicly known function that can be quickly computed.

The hash function must satisfy certain properties. The most common assumption is that it is "collision-resistant."

This means that there is no feasible way to find any pair of messages m, m' with the same hash value $H(m') = H(m)$.

In reduction arguments it is often very helpful to make a stronger assumption about H — that it behaves as a random function.

In other words, instead of assuming that there's a (publicly available) deterministic algorithm to compute H, one supposes that any time someone wants a hash value $H(m)$ an "oracle" will give her a random value to serve as $H(m)$ — with the only condition being that if the same m is given to the oracle again, then the oracle must answer with the same $H(m)$.

If we adopt this assumption, we are said to be working "in the random oracle model."

The basic RSA signature:

We'll suppose that the hash function H takes (for all practical purposes random) values $0 < H(m) < n$, where n is Alice's modulus.

Alice wants to sign a message m that she's just sent to Bob.

She finds $H(m)$ and then exponentiates mod n using her secret exponent d; that is, her signature is

$$s = H(m)^d \pmod{n}.$$

When Bob gets the message m and the signature s, he too computes $H(m)$, and then he uses the publicly known exponent e to verify that

$$H(m) = s^e \pmod{n}.$$

If this holds, then Bob is satisfied that

• Alice truly sent the message (because presumably only Alice knows the exponent d that inverts the exponentiation $s \mapsto s^e$ mod n); and

• the message has not been tampered with (because no one would be able to come up with a second message m' for which $H(m') = H(m)$).

The signature analogy of

> **"secure against chosen-ciphertext attack"**

is

> **"secure against chosen-message attack by an existential forger."**

This means that the adversary can get Alice to provide valid signatures s_i for any messages m_i of his choosing.

He will be judged to have broken the system (that is, successfully committed forgery) if he produces a valid signature for a message m that is different from all the m_i.

THEOREM. If the problem of inverting $x \mapsto x^e \pmod{n}$ is intractable, then the above RSA signature is secure (in the random oracle model) from chosen-message attack by an existential forger.

PROOF. We need to give a reduction argument — a reduction from inverting the RSA map to existential forgery. This means the following.

Suppose we have a computer program that takes as input Alice's public key (n, e). It is allowed to make a series of at most q queries for $H(m_i)$ and later (in some but not all cases) for the corresponding signature s_i for messages m_i of its choice.

To say that the computer program is a chosen-message existential forger means that it will eventually output a valid signature for one of the messages m for which it was given only the hash value $H(m)$ and not the signature.

We must show how such a computer program can be used to solve the problem: Given arbitrary y, find x such that $y \equiv x^e \pmod{n}$.

So suppose we are given such a y.

We give the computer Alice's (n, e) and wait for its queries.

In all cases but one, we respond to the hash query for m_i by randomly selecting x_i between 0 and $n-1$ and setting $H(m_i) = x_i^e \pmod{n}$.

For just one value m_{i_0} we respond by setting $H(m_{i_0}) = y$.

(Recall that y is the integer whose e-th root modulo n we need to find.)

Here we choose i_0 at random and hope that $m = m_{i_0}$ just happens to be the message whose signature will be forged by the existential forger.

Any time the forger asks for the signature of one of the m_i, $i \neq i_0$, we send x_i as its signature.

This will satisfy the forger, since $x_i^e \equiv H(m_i) \pmod{n}$.

If the forger ends up outputting a valid signature s_{i_0} for $m = m_{i_0}$, then we're done: we have a solution $x = s_{i_0}$ to $y \equiv x^e \pmod{n}$.

If we guessed wrong, which happens with probability at most $1 - 1/q$, then we won't get anything useful out of this run of the forger; so we run the program again.

If we go through the procedure k times, the probability that every time we fail to solve our congruence $y \equiv x^e \pmod{n}$ for x is at most

$$\left(1 - \frac{1}{q}\right)^k.$$

For large k this rapidly $\longrightarrow 0$.

So with high probability we succeed. QED

Informal version (without extraneous features):

Because H is random, the messages m_i are irrelevant.

What the forger has to work with:
a random sequence h_i
along with the corresponding x_i
(their e-th roots mod n)

To succeed, the forger must produce an e-th root mod n of a certain random h, $h \notin \{h_i\}$.

The theorem says that this is no easier than producing the e-th root mod n of a given random h without having the sequence of pairs (h_i, x_i).

The proof boils down to the following trivial observation:

Since both the h_i and the x_i are randomly distributed over the interval from 0 to $n-1$, you can obtain an equally good sequence of pairs (h_i, x_i) by starting with the random x_i and exponentiating with the publicly known exponent e, i.e., $h_i = x_i^e \pmod{n}$.

In other words, a sequence of random

$$(h_i, h_i^d \bmod n)$$

is indistinguishable from a sequence of random

$$(x_i^e \bmod n, x_i);$$

it makes no difference whether you look at your sequence of pairs left-to-right or right-to-left.

The above proof is really just the following tautology:

The problem of solving an equation is equivalent to the problem of solving the equation in the presence of some additional data (h_i, x_i) that are irrelevant to the problem and that anyone can easily generate using publicly available information.

But there is one subtlety that must be discussed.

In the above reduction proof the forgery-program must be run $O(q)$ times in order to be almost certain to find the desired e-th root of y modulo n:

$$\left(1 - \frac{1}{q}\right)^k \ll 1 \qquad \textbf{for} \quad k = O(q).$$

What are the practical implications (in the sense of Bellare–Rogaway's term "practice-oriented provable security")?

Suppose that you're using a large enough n so that you're confident that e-th roots mod n cannot be found in fewer than 2^{80} operations.

Suppose that you anticipate a chosen-message attack where the adversary will be able to make a million ($= 2^{20}$) signature queries.

The above reduction means that you can be sure only that the forger will require time at least

$$2^{80}/2^{20} = 2^{60}.$$

On the other hand, the informal version says that the forger will need the same amount of time — namely, 2^{80} — as anyone else who has to produce an e-th root modulo n.

Which is right,

(1) the reduction proof?

(2) common sense?

Equivalent question: Are the following two problems equivalent?

The RSA-problem:

> **Given n, e, y,**
> **find the e-th root of y mod n.**

The RSA1(q)-problem:

> **Given n, e and a finite set of random y_i, you are permitted to select up to q of those y_i, for each of which you will be given its e-th root mod n; you must produce the e-th root mod n for one of the remaining y_i.**

Informal argument:

> **RSA1(q) is no easier than RSA.**

Formal reduction:

> **RSA can be solved using $O(q)$ iterations of an algorithm for RSA1(q).**
>
> **So a lower bound for RSA must be divided by $O(q)$ to get a lower bound for RSA1(q).**

Many researchers developed modifications of the original RSA signature scheme that have "tight" reductions.

That is, they can prove that an algorithm to break their system can be used to solve the RSA-problem in essentially the same amount of time.

Is this worthwhile?

Is it wise to replace a simple system by a more complicated one in order to be able to give a "tight" reduction argument?

My opinion: No!

(1) The more complicated something is, the more that can go wrong with it. For example, if it requires a random number generator, then someone might find a way to break into that device.

(2) In this whole business everyone is implicitly assuming that the RSA-problem is as hard as factoring. By Boneh–Venkatesan, it is very unlikely that there's a reduction from factoring to the RSA-problem. So this assumption is based on experience over the years, "common sense," and the "gut feeling" of experts.

It's inconsistent to refuse to use "common sense" in the case of RSA vs RSA1(q).

In the presumed equivalence

$$\textbf{Factoring} \Longleftrightarrow \textbf{RSA} \Longleftrightarrow \textbf{RSA1}(q),$$

the first $\Longleftrightarrow$ is most likely the weakest link.

In other words:

Tight formal reduction arguments are nice to have.

But sometimes in cryptography one wants to assume that $\mathcal{P}_1$ is as hard as $\mathcal{P}_2$ even if there is no prospect of ever constructing such a reduction from $\mathcal{P}_2$ to $\mathcal{P}_1$.

Summary:

I. A "proof of security" establishes security only against a certain type of attack — and only conditionally on a certain problem being hard.

The Rabin example shows that the same feature that allows one to prove security against one type of attack could make the system totally vulnerable to another type of attack.

We're so busy improving the lock on the front door that we fail to notice that the back door is wide open.

II. A "proof of security" doesn't always agree with common sense.

The lack of a tight reduction argument might lead us to shun a simple system and prefer a more complicated one.

This might or might not be a good idea.

III. The "provable security" field lost credibility when the Bellare–Rogaway "proof" for their Optimal Asymmetric Encryption was found to be fallacious.

In the crypto research community there's an unfortunate tradition of trying to rush as many papers into print as rapidly as possible.

There's not much of a tradition of careful reading of other people's papers.

For 7 years no one examined the 1994 Bellare–Rogaway paper carefully, even though it was a very important paper that influenced real-world practice.

Notice the contrast with math:

1993 — examination of Wiles' FLT manuscript

2002 — 'Primes is in P' by Agrawal et al

IV. Most papers in the field are hard to read — math/comp.sci. jargon, excessive formalism.

V. The use of the theorem/proof paradigm is unfortunate.

I prefer the words "claim" and "argument"

(more precisely:

'reductionist security claim'

and

'reduction argument to support the claim').

Two things wrong with the word "proof" here:

(1) it connotes 100% certainty;

(2) it suggests an intricate, highly technical sequence of steps, which no one outside an elite of narrow specialists is likely to understand or raise doubts about.

A "proof of a theorem" is an intimidating notion.

An "argument in support of a claim" sounds humbler, and it suggests something that any well-educated person can try to understand and perhaps question.

Conclusion:

In the U.S. there's a long tradition of over-hyping the power of technology to guarantee security.

In the early decades of the Nuclear Age:

fallout shelters, bomb drills.

More recently:

demagogic politicians spending billions of dollars to construct "missile shields".

In cryptography, as elsewhere, we shouldn't fool ourselves into a false sense of security.

Does provable security exist?

My answer: No.

The search for secure cryptosystems is more an art than a science, and we can never be 100% sure that we've achieved success.