Irreducibility to the One-More Evaluation Problems: More May Be Less

Daniel R. L. Brown*

June 9, 2010

Abstract

For a random-self-reducible function, the evaluation problem is irreducible to the one-more evaluation problem, in the following sense. An irreduction algorithm exists that, given a reduction algorithm from the evaluation to the one-more evaluation problem, solves a separator problem: the evaluation problem itself. Another irreduction shows that if the computational Diffie-Hellman problem is reduced to the gap Diffie-Hellman problem, then the decision Diffie-Hellman problem is easy. Irreductions are primarily of theoretical interest, because they do not actually prove inequivalence between problems. What these irreductions suggest, though, is that one-more variants of the RSA and discrete logarithm problems may be easier than the standard variants, and that the gap Diffie-Hellman problem may be easier than the standard Diffie-Hellman problem.

1 Preliminaries

The *evaluation problem* for a function $f : X \to Y$ is to compute $f(x)$ given x, where x is chosen uniformly at random from X. The following are examples of cryptographically useful functions f whose evaluation problem is deemed to be difficult, at least for an adversary. Evaluating this first function is the RSA problem.

Definition 1. *Let n be a large integer, whose factorization is secret. (More specifically, $n = pq$, where p and q are sufficiently large random primes.) Let e be a fixed integer with $\gcd(e, \phi(n)) = 1$. Let $X = Y = (\mathbb{Z}/n)$. Let $f = \mathsf{RSA}$ be the function $f : X \to X : x \mapsto x^{1/e} \bmod n$. (The integers n and e are public.)*

The RSA problem has a trapdoor: an efficient algorithm to compute the function. This efficient algorithm requires access to secret information, which is not available to the adversary. The evaluation problem is defined for the adversary who is only given public information. It is commonly conjectured, though unproven, that the RSA problem is as hard as factoring integers.

Evaluating the next, second function is the discrete logarithm problem.

Definition 2. *Let $X = \mathbb{G}$ be a group of prime order n, written with multiplicative notation. Let $Y = \mathbb{Z}/n$. Let $g \in \mathbb{G}$ be fixed. Let $f = \mathsf{DL}$ be the function such that $f : X \to Y : g^x \mapsto x$.*

*Certicom Research

This function does not have a known trapdoor, unlike the RSA function. Therefore it may be difficult for anybody to solve this problem, not just an adversary. The discrete logarithm problem is conjectured to be hard for certain groups $\mathbb{G}$, such as certain subgroups of multiplicative groups of finite fields, and certain subgroups of elliptic curve groups over finite fields.

Evaluating the following, third function is the (computational) Diffie-Hellman problem.

Definition 3. *Let $X = \mathbb{G}$ be a group of prime order n, written with multiplicative notation. Let $g \in \mathbb{G}$ be fixed. Let $f = \mathsf{DH}$ be the function such that $f : X \times X \to X : (g^a, g^b) \mapsto g^{ab}$, where $a, b \in \mathbb{Z}/n$.*

This function, like the previous, has no known trapdoor. Various results indicate that its hardness is closely related to the discrete logarithm problem. Unlike the first two problems, the function is not injective. To be general enough to accommodate this problem, we formulate the conjecturally hard problem as function *evaluation* problems, not as function *inversion* problems, as is sometimes done (for example, see [1]).

The following is related to the strong (or static) Diffie-Hellman problem.

Definition 4. *Let $X = \mathbb{G}$ be a group of prime order n, written with multiplicative notation. Let $s \in \mathbb{Z}/n$ be a secret integer. Let $f = \mathsf{SDH}$ be the function such that $f : X \to X : x \mapsto x^s$. (Public information that may be used to attempt evaluate this function may include X and n and perhaps (g, g^s).)*

This problem is a variant of the previous. Like the RSA problem, this problem has a trapdoor: namely the secret s that permits efficient evaluation of the function. If this secret is not given to an adversary, then it becomes a hard problem for the adversary to evaluate. If one can evaluate DH then one can evaluate the SDH. The converse would be true if one had an algorithm that solves SDH for any random choice of s. However, in this problem the choice of s is not defined to be uniformly random over X, but rather only to have sufficient randomness to make it secret from an adversary. (This is what may be called the static Diffie-Hellman problem.) This is a rather a subtle distinction, which may be ignored, if desired. Once we formulate the one-more evaluation problem for this function, we get a form of what is known as the *strong* Diffie-Hellman problem, which has attracted greater interest since certain protocols require its difficulty and can be proved secure assuming its difficulty.

Random-self-reducibility is a property known to hold for various function used in cryptology, including those listed above. We need this property to establish our main results in this paper.

Definition 5. *A function $f : X \to Y$ is random-self-reducible, if there is an efficiently sampleable family of pairs of deterministic algorithms $(a_k : X \to X, b_k : Y \to Y)$ such that*

1. *$f = b_k \circ f \circ a_k$, and*
2. *For random index k and fixed $x \in X$, the random variable $a_k(x)$ is indistinguishable from a uniform random variable from X.*

In other words, blinded evaluation is possible. If you wish to evaluate f on a value x using an oracle for evaluating f, but do not wish to reveal any information about x to the oracle, then this is possible. Functions (a_k, b_k) for the four problems above are fairly easy to find. More generally, for any problem P, we can define it to be random-self-reducible if any instance can be transformed into another random instance whose solution can be used to solve the original instance.

Cryptology, including both cryptanalysis and provable security, often makes use of reductions between problems.

Definition 6. *Let P and Q be problems. The notation $P \Rightarrow Q$ means that there is an algorithm R, called a reduction, that, given an oracle for an algorithm A that solves problem P, can solve problem Q. As needed, we parameterize the reduction R by its computational cost, by the number of times it queries its oracle for A, by the parameters of the problems P and Q, and by its probability of success, especially as a function of the probability of success of its oracle A.*

When $P \Rightarrow Q$, one says that problem Q reduces to problem P, and the algorithm R is a reduction from Q to P (note that the "to" is opposite in direction to the arrow). In (conditional) cryptanalysis, P might be a problem such as integer factorization, while Q might be the problem of breaking a public-key encryption scheme. In provable security, the corresponding goal is to reverse the P and Q. An alternative notation is $P \geq Q$, which conveys the impression that P is at least as hard as Q.

Bellare, Namprempre, Pointcheval and Semanko [1] first introduced the one-more evaluation problems. Paillier and Vergnaud [7] also define one-more evaluation variant of the discrete logarithm problem.

Definition 7. *The one-more evaluation problem for a function $f : X \to Y$ is the problem of evaluating f on $t+1$ uniformly random given inputs $x_0, \ldots, x_t$ given an oracle that evaluates f on t arbitrary chosen outputs $y_1, \ldots, y_t$. If t is an input to the problem, then the problem is a closed variant, t-OM-f. If t is selected by the problem solver, the problem is the open variant, $*$-OM-f. When clear from context, the problem 0-OM-f may be denoted simply by f.*

In this notation, m-OM-RSA is the problem called RSA-KTI[m] in [1]; $*$-OM-RSA is problem called RSA-AKTI in [1]; and t-OM-DL is the problem called t-DL in [7]. Although the following reduction between one-more evaluations may be obvious we include a proof for completeness.

Lemma 1. *If f is any function, then t-OM-$f \Rightarrow (t+1)$-OM-f.*

Proof. The reduction R solves $(t+1)$-OM-f by calling an oracle A for the problem t-OM-f and then using its extra leftover f evaluation to evaluate the function f one more time than A can. In detail:

1. Reduction R is given $x_0, \ldots, x_{t+1}$ as its inputs to the $(t+1)$-OM-f.
2. Reduction R invokes A that with inputs $x_0, \ldots, x_t$.
3. Oracle A queries its f evaluation oracle with t inputs $y_1, \ldots, y_t$.
4. Reduction R queries its f evaluation oracle with inputs $y_1, \ldots, y_t$ and returns the answers back to A.
5. Algorithm A succeeds and returns $f(x_0), \ldots, f(x_t)$.
6. Reduction R has called its f evaluation oracle t times, but is permitted one more call, which it makes with input x_{t+1}.

Reduction R calls A exactly once. The computational cost of R is just bookkeeping. The success probability of R is the same as that of A. □

2 Irreductions

An *irreduction* is a reduction showing that finding another reduction is hard. There are several known irreductions in cryptology (see below), but the actual term irreduction has not usually been used. The formal definition is as follows.

Definition 8. *Let P, Q, S be problems. Then $P \not\Rightarrow_S Q$ means that there exists an algorithm M that, given an oracle for a reduction R showing $P \Rightarrow Q$, can solve problem S. Algorithm M is said to be an irreduction. Problem Q is said to be irreducible to problem P via the separator problem S. If $Q = S$, then we say that the irreduction is optimal, and write $P \not\Rightarrow_! Q$.*

An alternative notation for the irreduction $P \not\Rightarrow_S Q$ is $(P \Rightarrow Q) \Rightarrow S$, where the parentheses express the problem of finding the reduction $P \Rightarrow Q$.

Certain results may be expressed concisely with this notation.

- Bellare, Namprempre, Pointcheval and Semanko [1] proved that $\mathsf{RSA} \not\Rightarrow_{*\text{-}\mathsf{OM\text{-}RSA}} \mathsf{IF}$, where IF is the integer factorization problem. This is not an optimal irreduction.
- Boneh and Venkatesan [2] proved that $\mathsf{RSA} \not\Rightarrow_{\mathsf{IF}}^{\text{Alg}} \mathsf{IF}$, where RSA is the low exponent RSA problem and IF is integer factorization, and the superscript Alg indicates that irreduction only applies to algebraic reductions. This would be an optimal irreduction, but for the fact that it is limited to algebraic reductions. (If it were optimal, it would subsume the previous irreduction.)
- Paillier and Vergnaud [7] proved $\mathsf{Schnorr} \not\Rightarrow_{*\text{-}\mathsf{OM\text{-}DL}}^{\text{Alg}} \mathsf{DL}$, where $\mathsf{Schnorr}$ is problem of finding a universal forgery under a key only attack on Schnorr signature scheme, and DL is the discrete logarithm problem, and $*\text{-}\mathsf{OM\text{-}DL}$ the corresponding one-more evaluation problem. This is not an optimal irreduction.

The following is the main result of this paper. An almost identical[1] result was found independently by Bresson, Monnerat and Vergnaud [3].

Theorem 1. *If f is random self-reducible, then*

$$(t+1)\text{-}\mathsf{OM}\text{-}f \not\Rightarrow_! t\text{-}\mathsf{OM}\text{-}f. \tag{1}$$

Proof. Irreduction M for (1) work, at a high level, as follows.

1. Irreduction M is given inputs $x_0, \ldots, x_t$ to the problem t-OM-f. It invokes an oracle for reduction algorithm R with the same inputs.
2. The reduction algorithm R invokes an oracle for an algorithm A that solves $(t+1)$-OM-f, with inputs $y_0, \ldots, y_{t+1}$.
3. Reduction algorithm R can send inputs $u_1, \ldots, u_t$ to its oracle for evaluation by f. Irreduction M forwards these inputs its oracle for evaluating f and returns the answers back to R.

[1] Minor differences include: they do not limit the problem to an evaluation problem; they limit the problem to be one whose solution is easily verifiable; they also deal with the issue of "rewinding" reductions, such as used in the "forking lemma", whereas this is implicitly ignored here.

4. Irreduction M samples indices $k_0, k_1, \ldots, k_t$, and computes $w_i = a_{k_i}(x_i)$. Then M asks, in the role of A, the oracle for the reduction algorithm R to evaluate $f(w_i)$ for each i.

5. If R returns to A the answers $q_i = f(w_i)$ for all $i \in \{0, \ldots, t\}$, then M returns $z_i = b_{k_i}(q_i)$ as its evaluations of $f(x_i)$ for all $i \in \{0, \ldots, t\}$. Although M will not be able to complete the simulation of A by solving the task that A was given, this does not matter because M has already accomplished its task.

6. If R does not return the evaluations q_i, then M does not return the evaluations of f upon the inputs $y_0, \ldots, y_{t+1}$ to A.

7. If R evaluates f on all $t+1$ of inputs, namely $f(x_0), \ldots, f(x_t)$, then M returns the same answers as its solution.

If reduction R invokes its oracle A multiple times, then, in each invocation of A, irreduction M simulates A as described above. The only computation that M needs is the selection of k_i, evaluation of the functions a_{k_i} and b_{k_i}, and general bookkeeping. Irreduction M only invokes its oracle R exactly once. The probability of M succeeding is at least that of R.

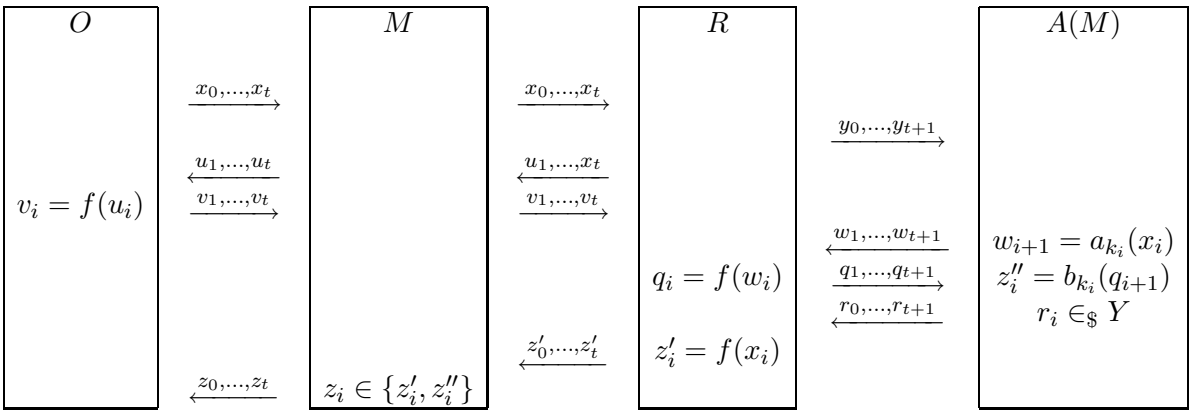

Figure 1: Irreduction M for $(t+1)$-OM-$f \not\Rightarrow_! t$-OM-f

If we allow R to return false answers to the queries of A, then we reply with random (false) answers as the solution to A. For some choices of f, false answers are easily detectable. For other choices of f, however, the problem of detecting whether evaluations of f are correct is also deemed to be a difficult problem in its in own right. If this is the case, then we slightly modify the strategy and the probability of success for M. Randomly select in advance whether the strategy of M is to use one of the intermediate values that R provides to A, or the final answer that R gives to its challenge. Suppose that the probability of success of R is ρ and the number of times that R invokes its oracle for A is α. Suppose further that ρ_1 is the probability that R is successful but it never completes a session with an A-oracle in which it answers all the A queries correctly. Let $\rho_2 = \rho - \rho_1$, which is the probability that R is successful but in at least one session with an A oracle, reduction R answers all queries from A correctly. The probability of success of M is $(\rho_1 + \rho_2/\alpha)/2$. We note that in this case, the irreduction M can be considered to be loose. □

This irreduction M implies an irreduction N for u-OM-$f \not\Rightarrow_! t$-OM-f, whenever $u > t$. To see this, note that there is a reduction R for $(t+1)$-OM-$f \Rightarrow u$-OM-f, which is obtained by iterating the reductions from Lemma 1. Irreduction N imitates M, except that whenever M deals with $(t+1)$-OM-f, instead N deals with u-OM-f and converts this via reduction R to $(t+1)$-OM-f. In particular, we have t-OM-$f \not\Rightarrow_! f$ for all $t > 0$. In words, the evaluation problem is optimally irreducible to each of the one-more evaluation problems.

A similar result for the open variant of the one-more evaluation problems is the following.

Theorem 2. *If f is random self-reducible, then*

$$*\text{-OM-}f \not\Rightarrow_! t\text{-OM-}f. \tag{2}$$

Proof. Irreduction M works as follows.

1. Irreduction M is given inputs $x_0, \ldots, x_t$ on which it is to evaluate the function f.
2. Irreduction M invokes oracle R that solves the t-OM-f using an oracle A for solving $*$-OM-f. The input that M provides to R is its own point of $x_0, \ldots, x_t$.
3. Reduction R has access to another oracle that will evaluate f on up to t elements of X. This is because, R is trying to solve the t-OM-f. To answer these queries, irreduction M invokes its own oracle for evaluating f, which it too can invoke up to t times.
4. When R invokes its oracle A for solving $*$-OM-f, the irreduction M provides a simulation of A. When M simulates A, it chooses to make $t' = t+1$ invocations to its f-evaluation oracle to evaluate f at $t'+1$ given inputs, which are selected by R, with values, say, $z_0, z_1, \ldots, z_{t+1}$.
5. To help it in its task, A is allowed access to an f-evaluation oracle that it may call t' times, where t' is the value above. The queries that M, in its role as A, makes to R are for evaluations of f at randomization of its challenges $x_0, \ldots, x_t$. Say $x'_i = a_{k_i}(x_i)$ for k_i chosen at random.
6. If R correctly evaluates $y'_i = f(x'_i)$, then M can solve its challenges correctly with $y_i = b_{k_i}(y'_i)$. It is not necessary for M to actually provide the answers to the problem that A is to solve, because now M has everything it needs to solve its own problem.
7. Otherwise, R incorrectly evaluates f on its queries from A, then A is not obliged to solve $*$-OM-f, so M need not provide a correct answer to R.
8. In this case, R will solve its given challenge, which is also a solution of the challenge given to M.

Irreduction M invokes R once. The cost of M is bookkeeping plus the cost of selecting and evaluating the functions a_k, b_k.

If f is verifiable, then the probability of success of M is the same as that of R. If f is not easily verifiable, then M employs a probabilistic strategy, and has probability of success of $(\rho_1 + \rho_2/\alpha)/2$, where ρ_1 and ρ_2 is the probability of success of R, when R is always dishonest and someimtes honest, respectively, and α is the number of times that R invokes A. □

Another type of irreduction is now given. In some cases, the problem of verifying evaluation function f is deemed difficult and is called the *decision problem* for f, which we write as DE-f. Formally, for a function $f : X \to Y$, the decision problem is, given (x, y), determine whether $y = f(x)$. The given (x, y) are distributed with x chosen uniformly at random from X. The value for y is chosen to be $f(x)$ with probability $\frac{1}{2}$, and otherwise is chosen uniformly at random from Y.

When the decision problem is deemed hard, it is also possible to formulate a *Gap* problem GE-f. This problem is to evaluate f given an oracle for the decision problem. In the case of the Diffie-Hellman function $f = \mathsf{DH}$, exponentiation by a secret exponent, then we have the Diffie-Hellman problem 0-OM-$f = \mathsf{CDH}$, the decision Diffie-Hellman problem DE-$f = \mathsf{DDH}$ and the gap Diffie-Hellman problem GE-$f = \mathsf{GDH}$. The GDH has attracted attention because it has proven possible to reduce it to breaking the security of several cryptographic schemes.

Theorem 3. *If f is random-self-reducible function, then GE-$f \not\Rightarrow_{\mathsf{DE}\text{-}f}$ 0-OM-f. In particular, $\mathsf{GDH} \not\Rightarrow_{\mathsf{DDH}} \mathsf{CDH}$*

Proof. The irreduction M has the following procedure.

1. Irreduction M is asked to solve DE-f. Its input takes the form (x, y) and M must determine whether $y = f(x)$.

2. Irreduction M invokes its oracle reduction R that solves the 0-OM-f using an oracle for solving GE-f. The input that M supplies to R is x.

3. Reduction R invokes its oracle A for solving GE-f, say with input z. In other words R asks A to compute $f(z)$.

4. Given that A is trying to solve GE-f, it is only required to succeed if R can provide a valid oracle that solves the DE-f. Irreduction M, in the role of A, asks R to solve the DE-f with input $(a_k(x), b_k(y))$. If R answers, then M returns this answer as its own solution to the DE-f.

5. If R does not solve the decision problem for A, then M does not evaluate $f(z)$ for R.

6. Now R has received its valid oracle A for solving GE-f, as simulated by M, so R, by hypothesis, can solve 0-OM-f, which in this case, means that R can compute $w = f(x)$.

7. Finally M verifies that $w = y$, and solves its given instance of DE-f accordingly.

Irreduction M calls R just once. The main cost of M is bookkeeping, and selection and evaluation of the random-self-reduction functions a_k and b_k. If R invokes A multiple times, M employs the same strategy.

In this case, the function f is presumed not to be easily verifiable, so that M employs a probabilistic strategy. The probability of success of M is $(\rho_1 + \rho_2/\alpha)/2$, where ρ_1 and ρ_2 is the probability of success of R, when R is always dishonest and someimtes honest, respectively, and α is the number of times that R invokes A. □

3 Conclusion

An *irreduction* algorithm shows that one cannot reduce certain cryptographically useful evaluation problems — such as the RSA problem, the discrete logarithm problem, or the Diffie-Hellman problem — to their one-more evaluation variant. If one is to *prove* that one of these one-more evaluation problem is as hard as the corresponding evaluation problem, then the proof technique must use something beyond a direct reduction algorithm. These irreductions may be interpreted as evidence that the one-more evaluation problem is easier than the single evaluation, although it is not a proof of such a gap.

Another irreduction algorithm shows that one cannot reduce the Diffie-Hellman problem to the Gap Diffie-Hellman problem, unless the Decision Diffie-Hellman problem is easy. This is evidence for, but not a proof of, a gap between the standard and gap Diffie-Hellman problems. It should noted, however, that this irreduction is loose.

4 Acknowledgments

I thank Alfred Menezes and Jean Monnerat for valuable comments.

References

[1] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-RSA-inversion problems and the security of chaum's blind signature scheme. *Journal of Cryptology*, 16(3):182–215, 2003.

[2] D. Boneh and R. Venkatesan. Breaking RSA may be easier than factoring. In K. Nyberg, editor, *Advances in Cryptology — EUROCRYPT '98*, number 1403 in LNCS, pages 59–71. IACR, Springer, May 1998. `http://crypto.stanford.edu/~dabo/abstracts/no_rsa_red.html`.

[3] E. Bresson, J. Monnerat, and D. Vergnaud. Separation results on the "one-more" computational problems. In T. Malkin, editor, *Topics in Cryptology — CT-RSA 2008*, LNCS. IACR, Springer, Apr. 2008. To appear.

[4] N. Koblitz and A. Menezes. Another look at non-standard discrete log and Diffie-Hellman problems. CACR 2007-03, University of Waterloo, Nov. 2007. `http://www.cacr.math.uwaterloo.ca/techreports/2007/cacr2007-32.pdf`.

[5] N. Koblitz and A. Menezes. Another look at "provable security". *Journal of Cryptology*, 20:3–37, 2007.

[6] H. Lipmaa. On CCA1-security of Elgamal and Damgård cryptosystems. ePrint 2008/234, IACR, May 2008. `http://eprint.iacr.org/`.

[7] P. Paillier and D. Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In B. Roy, editor, *Advances in Cryptology — ASIACRYPT 2005*, number 3788 in LNCS, pages 1–20. IACR, Springer, Dec. 2005.

A Lack of a Contradiction

Theorem 2 may seem to contradict [1, Theorem 5.3], whose proof implies the following. If $*$-OM-f can be solved efficient by algorithm A, then there exists an algorithm B and a t, at most a constant factor times the computational cost of A, such that B solves t-OM-f. This may appear to be saying that $*$-OM-$f \Rightarrow t$-OM-f, for some t, which combined with irreduction $*$-OM-$f \not\Rightarrow_! t$-OM-f above suggests that t-OM-f can be solved. This conclusion is false. The logic is correct, but the premise $*$-OM-$f \Rightarrow t$-OM-f is not correct. However, [1, Theorem 5.3] is also correct, but it was erroneous to write $*$-OM-$f \Rightarrow t$-OM-f, because the t on the right is fixed, whereas on the left the t is chosen by the problem solver. Formally, this may resolve the apparent contradiction, but a more intuitive explanation may help.

We look again at what is happening. In the irreduction above, we are given, say t-OM-f, and then we run an algorithm for solving $*$-OM-f by choosing $t' = t + 1$. The reduction in [1, Theorem 5.3] works by observing the choice of t' made in an algorithm A for solving $*$-OM-f, and then it solves t'-OM-f. Combining these results in an attempt to solve t-OM-f, we would be trying to solve t-OM-f by trying to solve $(t+1)$-OM-f. If we iterated this, then t would go to infinity, which does not give an efficient solution.

B On Optimality of Irreductions

One reason that, when $Q = S$, the irreduction $P \not\Rightarrow_S Q$ deserves to be called optimal is the following.

Lemma 2. *If $P \not\Rightarrow_S Q$, then $Q \Rightarrow S$.*

Proof. We convert the metareduction M for $P \not\Rightarrow_S Q$ into a reduction M' for $Q \Rightarrow S$. If A' is an algorithm that solves the problem Q, then it is also a reduction R for $P \Rightarrow Q$, that does not make use of any oracle for solving P. By definition, M can use R to solve S, so M' uses to A' to solve S. Because R does not use an oracle for solving P, the irreduction M does not need to simulate this oracle. Therefore, when M' imitates M exactly, it does not need to simulate an algorithm for solving P, therefore fulfills the definition of a reduction $Q \Rightarrow S$. □

Returning to the point of this lemma, it states that the separator problem S can be no harder than the problem being reduced. We now adopt an extremely heuristic notation. For problem P, let $|P|$ denote its difficulty, as measured, for example, by the computational cost of solving P. An irreduction $P \not\Rightarrow_S Q$, suggests that $|Q| - |P| \geq |S|$, and if this were true, then certainly $|Q| \geq |S|$. Finally, if the irreduction is optimal, then this heuristic further suggests that $|P| = 0$. It should be kept in mind that is purely heuristic, and that irreductions do not prove any difference in difficulty, and especially that optimal irreductions do not proved that the problem P is easy.

Because the separator problem S will never be harder than the reduced problem Q in an irreduction $P \not\Rightarrow_S Q$, the main task of the irreduction algorithm will be to simulate the oracle A for solving P that the reduction R expects. Indeed, if M can simulate A easily, then R will solve Q, and the irreduction will be optimal. In some proofs, R may not complete the task Q, because M can only partially simulate A. However, partial simulation of A may suffice to evoke R, as it participates in the game that A is trying to win, to provide information to M that it can use to

solve S. When the irreduction M uses such partial simulation of A it may be the case that S is an easier problem than Q.

Another reason for the definition of optimal irreductions is the following.

Lemma 3. *If $P \not\Rightarrow_S Q$ and $S \Rightarrow T$, then $P \not\Rightarrow_T Q$.*

Proof. Let M be the irreduction for $P \not\Rightarrow_S Q$ and let R be the reduction for $S \Rightarrow T$. We seek to construct an irreduction N for $P \not\Rightarrow_T Q$. By definition, N is given an instance of T to solve. We invoke R with the same instance of T. Whenever R invokes its S-oracle, we invoke M, with the same instance of S to solve. Whenever M invokes its $(P \Rightarrow Q)$-oracle available to it, we invoke the $(P \Rightarrow Q)$-oracle available to N. Therefore, M will solve S, and therefore R will solve T, and N has its desired solution. □

C Intuitionistic Tautologies and Generalized Metareductions

Lemma 2 may be re-written as:

$$((P \Rightarrow Q) \Rightarrow S) \Rightarrow (Q \Rightarrow S). \tag{3}$$

Let T be the reduction that was described in the proof. To describe such things involving several arrows and parentheses, we use the term *metareduction*. Metareductions are reductions, but we use this term when we want to emphasize the fact that the reduction has another metareduction on one or both sides. Irreductions are examples of metareductions. In previous work, irreductions have been called metareductions. Here, metareductions refers to something even more general.

The metareduction (3) is a *tautologous* metareduction in the sense that it is works no matter what the problems P, Q, and S are. The metareduction in Lemma 3 is also tautologous. If we view (3) as an expression in boolean variables, it is always true, making it a formal boolean tautology. We conjecture below that the converse is not true: not every boolean tautology gives rise a tautologous metareduction.

The metareductions above only use the logical operator of implication. (That in Lemma 3 is stated with a verbal conjunction operator, but can expressed using impliation operators only, for example, as $((S \Rightarrow T) \Rightarrow ((P \Rightarrow Q) \Rightarrow S)) \Rightarrow ((P \Rightarrow Q) \Rightarrow T)$.) No negation operator is used. Indeed, it is not clear what the negation of a problem should be. This subset of logic that we are using is known as *implicational propositional logic*. A theorem of Peirce states that any tautology in this logic can be deduced from the following three axioms:

$$\alpha(P, Q) = (P \Rightarrow (Q \Rightarrow P)) \tag{4}$$

$$\beta(P, Q, R) = ((P \Rightarrow (Q \Rightarrow R)) \Rightarrow ((P \Rightarrow Q) \Rightarrow (P \Rightarrow R))) \tag{5}$$

$$\gamma(P, Q) = (((P \Rightarrow Q) \Rightarrow P) \Rightarrow P) \tag{6}$$

The first two are due to Lukasiewicz, while the third is known as Peirce's law. By *deduction* above, we mean that the variables above may be substituted by arbitrary expressions (tautologous or not), to get larger tautologous expressions; and *modus ponens* may be used to get smaller expressions, which means that if T and $S = (T \Rightarrow U)$ are shown to be tautologies, then U is a tautology.

If we write $U = T \setminus S$, then we have a convenient notation for expressing deductions. If S and T are tautologies and T is the predicate of the S, then $T \setminus S$ is defined and is also a tautology.

The subset of tautologies that can be deduced from the first two axioms α and β are known as *intuitionistic*. Peirce's law is not intuitionistic. The Curry-Howard isomorphism between tautologies and programs in the λ calculus, is only defined on intuitionistic tautologies. In Heyting algebras, truth values can take on a larger set of values, and a value is an intuitionistic tautology if and only if the proposition evaluates to the maximally true value no matter what truth values the variables are assigned.

An example of an intuitionistic tautology is $P \Rightarrow P$, as the following deduction shows:

$$(P \Rightarrow P) = (\alpha(P, Q) \setminus (\alpha(P, Q \Rightarrow P) \setminus \beta(P, Q \Rightarrow P, P))) \tag{7}$$

Although this deduction may not be very intuitive, it does demonstrate that $P \Rightarrow P$ is intuitionistic.

Now returning to the tautologous metareductions, the *modus ponens* operation corresponds to composition of the algorithms for solving a problem. If algorithm A solves problem T and algorithm B solves problem $S = T \Rightarrow U$, then problem U may be solved by using B and invoking A whenever B invokes its oracle for solving T.

So, if we can find metareductions for the Lukasiewicz axioms, then we can compose these metareductions to find metareductions for any expression that corresponds to a intuitionistic tautology. The metareduction for $\alpha(P, Q)$ is quite simple. Given a P-solver A, one gets a $Q \Rightarrow P$ solver by invoking A, and not making an oracle queries at for the Q solver. Let us call this metareduction $M_\alpha(P, Q)$.

A metareduction M for $\beta(P, Q, R)$ works as follows. Suppose that A is a metareduction for $P \Rightarrow (Q \Rightarrow R)$. Essentially, this means that A solves problem R using oracles for problems P and Q. The first oracle for P is by definition. The second oracle, for Q, is from the intermediate problem $Q \Rightarrow R$ that A is trying to solve using a P-oracle. What M needs to do is to solve R using oracles for P and for $(P \Rightarrow Q)$. To do this, M call A to solve R. When A calls its P-oracle, then M forwards this to its P-oracle. Then A calls its Q oracle, then M calls its $(P \Rightarrow Q)$-oracle. This $P \Rightarrow Q$-oracle expects access to another oracle, a P-oracle. These queries are forwarded by M to its own P-oracle. Accordingly, the $P \Rightarrow Q$ oracle solves the given problem Q, which allows M answer the Q-oracle query for A. Therefore M can answer all the queries of A, so A will eventually solve problem R. Let us call this metareduction $M_\beta(P, Q, R)$.

As a simple example we apply this process to the trivial reduction $P \Rightarrow P$, which does have a intuitionistic deduction: (7). To do this, we use the metareduction $M_\beta(P, Q \Rightarrow P, P)$ for some problem Q. The problem that this metareduction solves requires an oracle to a problem which the metareduction $M_\alpha(P, Q \Rightarrow P)$ solves. The resulting composition of metareductions requires a further oracle to the problem which $M_\alpha(P, Q)$ solves. What we would like is that this rather complex composition of metareduction reduces to the trivial reduction.

Now the metareduction $M_\beta(P, Q \Rightarrow P, P)$ ultimately tries to solve P, perhaps by using some oracle queries. As we described M_β, it invokes $M_\alpha(P, Q \Rightarrow P)$, and handles its oracle queries by invoking its own oracles. The metareduction $M_\alpha(P, Q \Rightarrow P)$ tries to solve P using a P-oracle, and declines to use its available $(Q \Rightarrow P)$-oracle. Therefore, the only oracle queries from $M_\alpha(P, Q \Rightarrow P)$ is to the P-oracle. Now the axiomatic metareduction $M_\beta(P, Q \Rightarrow P, P$ essentially has a P oracle, and answers any P-oracle queries from $M_\alpha(P, Q \Rightarrow P)$ by invoking its P-oracle. The net result is that the available P-oracle is to used to solve the problem P, so the resulting reduction $P \Rightarrow P$ is the trivial reduction that we would hope for. (If $M_\alpha(P, Q)$ metareduction ever gets called too, then we note that it only solves P by invoking a P-oracle, so again nothing more complicated than the trivial reduction will be obtained.)

We have shown that every intuitionistic tautology has a metareduction. Perhaps, though, some non-intuitionistic tautologies have metareductions. A first example to consider is Peirce's law. We may re-write this law using our irreduction notation, in two different ways:

$$(P \not\Rightarrow_P Q) \Rightarrow P \tag{8}$$

$$(P \Rightarrow Q) \not\Rightarrow_! P \tag{9}$$

When expressed in this way, neither seems very likely. For example, we found an irreduction $\mathsf{1\text{-}OM\text{-}DL} \not\Rightarrow_! \mathsf{DL}$, which is optimal, and would therefore seem to be stronger than irreduction $\mathsf{1\text{-}OM\text{-}DL} \not\Rightarrow_{\mathsf{1\text{-}OM\text{-}DL}} \mathsf{DL}$. If Peirce's law has a metareduction, then we could use this latter irreduction to solve 1-OM-DL. More generally, Peirce's law suggest that Q is a problem strictly harder than P, then P is an easy problem, which is a form of the *law of the excluded middle*, which does seem to be a reasonble law for the hardness of problems.

This suggests the following conjecture.

Conjecture 1. *An expression has a metareduction if and only if it is an intuitionistic tautology.*

We already proved the if part, so only the only if part is conjectural. Failure to find a metareduction for Peirce's law would seem very little basis for this conjecture. As additional support, we suggest that there may be Heyting algebra here. In this case, truth values are the difficulty of the problems, and easy corresponds to maximally true.

Some further tests and applications for this conjecture are as follows. Recall that we found a metareduction for $((P \Rightarrow Q) \Rightarrow S) \Rightarrow (Q \Rightarrow S)$. If this expression is not an intuitionistic tautology, then the conjecture is wrong. A necessary[2] condition for an intuitionistic tautology is for to be $[0,1]$-tautology. In $[0,1]$-logic, variables may take on truth values for an real number in the interval $[0,1]$. In this logic, for truth values p and q, we define

$$p \Rightarrow q = \begin{cases} 1 & \text{if } p \leq q \\ q & \text{if } p > q \end{cases} \tag{10}$$

A $[0,1]$-tautology is any expression that evaluates to 1 for all truth value assignments of its variables. It is a routine verification to see that $((P \Rightarrow Q) \Rightarrow S) \Rightarrow (Q \Rightarrow S)$ is $[0,1]$-tautology. Peirce's is not a $[0,1]$-tautology, which can be seen by assigning $P = \frac{1}{2}$ and $Q = \frac{1}{3}$, which causes the law to evaluate to $\frac{1}{2}$. It is routine to verify that the Lukawiewicz axioms and *modus ponens* hold in $[0,1]$-logic, so therefore every intuitionistic tautolgys is a $[0,1]$-tautology.

As an application, we return to the question of whether our proof that $\mathsf{GDH} \not\Rightarrow_{\mathsf{DDH}} \mathsf{CDH}$ may have been stronger than necessary. For example, it depended on certain problems being random-self-reducible. To see this, we note that another way to express the Gap Diffie-Hellman problem is as:

$$\mathsf{GDH} = (\mathsf{DDH} \Rightarrow \mathsf{CDH}), \tag{11}$$

because the Gap Diffie-Hellman problem may be viewed as the problem of reducing the (computational) Diffie-Hellman problem to the Decision Diffie-Hellman problem. Suppose that we can find a tautologous metareduction M that shows:

$$(Q \Rightarrow P) \Rightarrow (((P \Rightarrow Q) \Rightarrow Q) \Rightarrow P). \tag{12}$$

[2]Is it also sufficient?

Then substituting $Q = \mathsf{CDH}$ and $P = \mathsf{DDH}$, the right term of (12) is the irreduction $\mathsf{GDH} \not\Rightarrow_{\mathsf{DDH}} \mathsf{CDH}$, and the left term is $\mathsf{CDH} \Rightarrow \mathsf{DDH}$, which is obvious. If (12) is an intuitionistic tautology, then it our proof was overly strong. However, if we can show that this is expression is not an intuitionistic tautology, then our conjecture would imply that even a strategy of using (12) for a proof would have to depend on some particular properties of the problems P and Q, such as random-self-reducibility. It is not too hard to find $[0, 1]$ truth value assignments for the varitables P, Q, R that make (12) evaluate to less than 1. Therefore, (12) is not a $[0, 1]$-tautology and therefore not an intuitionistic tautology.

D Singular and Robust Reductions, and Power Problems

Let P be a problem. Let P^t be the problem of solving t independent instances of P. A reduction for $P \Rightarrow Q$ is *singular* if it only invokes its P-oracle once. In this case, we write $P \Rightarrow_1 Q$. We may say that Q is singularly reducible to P. With these two notions, we may formulate the one-more variant of any problem P:

$$t\text{-}\mathsf{OM}\text{-}P = (P^t \Rightarrow_1 P^{t+1}) \tag{13}$$

A reduction R for $P \Rightarrow Q$ is *robust* if R succeeds even if the P-oracle given to it is not independent of the instance of the problem Q that R has to solve. In this case, we write $P \Rightarrow_\$ Q$. We may say that Q is robustly reducible to P. For example, a reduction R for $P \Rightarrow_\$ P$ may be given a P-oracle that fails for the instance of P that R is asked to solve, and therefore our trivial reduction $P \Rightarrow P$ that holds for all problems (by invoking its P-oracle with the same instance it is given) is not robust. If a problem is random-self-reducible, as we have defined it, then $P \Rightarrow_\$ P$. Because the converse may not hold, we say that P is robustly-self-reducible if $P \Rightarrow_\$ P$.

With these three notions, we formulate a conjecture that strengthens Theorem 1.

$$(P \Rightarrow_\$ P) \Rightarrow \big(\big(\big(P^{t+1} \Rightarrow_1 P^{t+2}\big) \Rightarrow \big(P^t \Rightarrow_1 P^{t+1}\big)\big) \Rightarrow \big(P^t \Rightarrow_1 P^{t+1}\big)\big) \tag{14}$$

By this, we mean that this metareduction (involving specialized reductions) should hold for any problem P. If so, it is a tautologous metareduction, in line with our earlier strategy of determining which metareductions are tautologous.

It remains to be seen whether this formulation provides any actual benefit. Even if a general calculus of such specialized reductions may be formed to prove (14), it is unclear if such a calculus would find any other applications in cryptology. If not, developping such a calculus may be more trouble than its worth.

E Koblitz-Menezes Problems

Koblitz and Menezes [5] defined a variant of the RSA problem that they called RSA1. They did this on the basis of work done by Coron towards the impossibility of a tight reduction for the security of RSA-FDH. Our previous notations do not express the notion of a tight reduction, but we now add this to our toolbox.

A reduction R for $P \Rightarrow Q$ is *lossless* if its probability of success is at least the probability of success of its P-oracle. In this case, we write $P \Rightarrow_+ Q$, and say that Q reduces losslessly to P. If a reduction is not lossless, then we may say that it is lossy. (Loosely speaking, a reduction is tight if it lossless and singular, which we write as $P \Rightarrow_{1,+} Q$.)

Koblitz and Menezes [4] have generalized the problem RSA1 to others, such as DLP1. Given a problem P, the Koblitz-Menezes variant t-KM-P of the problem is defined as follows. One is given $t+1$ instances of the problem to solve. One is given an oracle that will solve any of these problems. If one solves all of the problems, but invokes the problem-solving oracle at most t times, then one has solved the t-KM-P. This problem is a restriction of t-OM-P in the sense that the P-oracle will not solve arbitary instances of the problem P, but only those given to it. Because the problem solver in t-KM-P is just given a weaker oracle, the problem is potentially harder than t-OM-P. Indeed we have a fairly trivial reduction t-KM-$P \Rightarrow_{1,+} t$-OM-P, because if A solves t-KM-P then it solves t-OM-P.

Also obvious is the tight reduction $P \Rightarrow_{1,+} t$-KM-P, because we can call P-orcale given in t-KM-P to solve the first $t-1$ instances of P, and then call the P-oracle in the reduction. What really distinguishes the Koblitz-Menezes problem from the one-more problem is the following loose reduction, that works for a certain class of problems.

$$t\text{-KM-}P \Rightarrow P \tag{15}$$

The class of problems for which this reduction works is those that are *existentially* solvable, which means that there is an efficient algorithm that find pairs of problem and correspoding solution instances. Also, the problem instance produced will be indistinguishable from a uniformly[3] chosen problem instance.

The reduction R works as follows. It is given problem instance P_0 to solve. Choose a random integer $i \in [1, t+1]$ and set $P_i = P_0$. For the remaining P_j for $j \neq i$, exploit the existentially solvability of problem P to generate pairs of problem instances and solution instances (P_j, S_j). Now when R invokes an oralce A for solving t-KM-P with instance $(P_1, \ldots, P_{t+1})$. By definition of A, it does not query P for at least one of the problem instances, say P_k. With probability $\frac{1}{t+1}$, we will have that $k = i$. In this case, R can answer all the P-oracle queries from A, and therefore A will solve $P_i = P_0$ as R requires.

Actually, this is essentially the reduction that Bellare and Rogaway used to prove the security of RSA-DFH. Coron proved that the solving the RSA problem could not be tightly reduced to the security RSA-FDH. As Koblitz and Menezes observed, Coron effectively proved that RSA cannot be tightly reduced to RSA1. More generally, we may write this loosely as:

$$t\text{-KM-}P \not\Rightarrow_{1,+} P. \tag{16}$$

More formally, this can be expressed as irreduction:

$$(t\text{-KM-}P \Rightarrow_{1,+} P) \Rightarrow P \tag{17}$$

To see how to get such an irreduction M, we invoke the tight reduction oracle R twice, with the same instance of P to solve. We resort to a trick of fixing the random tape of R, which seems reasonable. This causes both instance of R, say R_1 and R_2 to invoke its t-KM-P oracle with the same problem instance, say $(P_1, \ldots, P_{t+1})$. Now, M asks R_1 to solve all problems except P_1 and M asks R_2 to solve all problem instances except P_2. Together M can solutions for all problem instance and can simulate the t-KM-P for both R_1 and R_2. Consequently R_1 will solve P and M is done. Again, this is the Coron's approach.

[3] Or with whatever probability distribution the problem instances are defined to take.

Note that this metareduction M does not require P to be existentially solvable, unlike the reduciton. Again, as part of a more general theory, it would be interesting to see to what axioms one might formulate to deduce such things. To this end, we may express

$$t\text{-}\mathsf{KM}\text{-}P = (P^t \Rightarrow_\sim P^{t+1}), \tag{18}$$

with the notation $\Rightarrow_\sim$ indicating the retstriction on the reduction that its oracle will only solve problems that are related in a certain manner to the problem instance that the reduction must solve. This type of reduction depends on the type of relation $\sim$. In the case of the Koblitz-Menezes problem, the $\sim$ is that the instance P^t must be a proper subset of the instance P^{t+1}. Note that $P \Rightarrow_\$ Q$ implies $P \Rightarrow_\sim Q$ implies $P \Rightarrow Q$, no matter what the relation $\sim$.

F Lunchtime Reductions

Koblitz and Menezes [4] identified another way to vary a given problem, which is once again similar to the one-more variant of a problem. They identified this as version of the "one-more" RSA problem addressed by Joux, Naccache and Thomé. Koblitz and Menezes called this the *delayed target* version of the one-more variant problem. However, this concept has also been used in defining adversaries to public key encryption schemes.

Let R be a reduction $P \Rightarrow Q$. If the reduction invokes its P-oracle before seeing its instance of the Q problem, and does not invoke its P-oracle after seeing its instance of the Q, it is a *lunchtime* reduction. In this case, we write $P \Rightarrow_< Q$. The delayed target version of the one-more variant of any problem P, may now be defined as

$$t\text{-}\mathsf{DT}\text{-}P = (P^t \Rightarrow_< P). \tag{19}$$

Joux, Naccache and Thomé showed that if P is the RSA problem, then this variant could be easier than P is conjectured to be. They did this by giving an algorithm for solving this problem that is faster than the best known algorithm for solving P. This is the more practical way to compare hardness of problems, which is often what an irreduction is said to suggest as possible. Here we make the converse suggestion. For example, is there an irreduction

$$t\text{-}\mathsf{DT}\text{-}P \not\Rightarrow_! P? \tag{20}$$

G Public Key Encryption

The previous section brings to mind the various security properties of a public key encryption scheme. In this section, we attempt to formalize these various notions in terms of the notations introduced above.

We now presume that we have some public key encrytion scheme Φ. Let f be the function that takes a valid ciphertext and returns in the corresponding plaintext. The evaluation problem for f obviously needs to be hard for Φ to be secure. Here we give the adversary the public key. This is the one of most basic security notions: *one-wayness against chosen plaintext adversaries*. We may write the problem for the adversary as $\mathsf{OW\text{-}CPA\text{-}}\Phi$. In other words $\mathsf{OW\text{-}CPA\text{-}}\Phi = f$.

We can also conside a more advanced security property as the problem:

$$\mathsf{OW\text{-}CCA2\text{-}}\Phi = (f \Rightarrow_\sim f) \tag{21}$$

where the relationship $\sim$ here is inequality. That is, we try to compute f using an f-oracle for any other intput. It has been argued that this security definition is too stringent, because it disallows something benign malleability. Therefore, we may define a potentially harder problem:

$$(f \Rightarrow_{\$} f) \tag{22}$$

Hardness of this problem is a milder security goal, and arguably a reasonable substitute for OW-CCA2 security. To model lunchtime chosen ciphertext attacks, we formulate the problem:

$$\mathsf{OW\text{-}CCA1\text{-}}\Phi = (f \Rightarrow_{<} f) \tag{23}$$

What about semantic security, or indistinguishability? This is something over and above one-wayness, and something that is usually considered a requirement for Φ. Earlier, for any evaluation problem f, we formulate the decision version DE-f. This is part of the way to what we want, becuase it presumes that if the adversary sees (x, y), where x is uniformly random, then the adversary cannot tell if $y = f(x)$. However, for indistinguishability, the adversary can choose a set $C \subset Y$ with as few as two elements, say $\{y_0, y_1\}$, and then constrain $f(x), y \in C$. This is the constrained decision problem CDE-f, which may be formulated for any function. Thus we may state:

$$\mathsf{IND\text{-}CCA2\text{-}}\Phi = (f \Rightarrow_{\sim} \mathsf{CDE\text{-}}f), \tag{24}$$

where $\sim$ means that the f-oracle is restricted to not return an answer for any query x with $f(x) \in C$. Again we posit that the milder security property of the problem $(f \Rightarrow_{\$} \mathsf{CDE\text{-}}f)$ being hard would generally be an adequate substitute for the hardness of IND-CCA2-Φ

Although the formulations attempt to capture the security properties of Φ in terms of reductins about variants of the evaluation problem f, they do not capture what is necessary about f for it to be usable in a public key encryption scheme. We want somebody to be able to decrypt ciphertexts, so the first essential property is that f must have a trapdoor, allowing some party to evaluate f, or else nobody could decrypt anything. (It should also be note that the problem must be an evaluation problem, if we want to have unique decryption.) On the other we want anybody to be able to encrypt, so we need a further property. If we wish to use Φ only a key encapsulation mechanism, as defined by Shoup, then it suffices f to existentially solvable, meaning that anybody can find pairs (x, y) such that $y = f(x)$, with the further property that y is has a uniform distribution over y (or at least sufficient entropy to derive a data encapsulation key). In the case of key encapsultion, it seems that the milder one-wayness form of security is sufficient, and that full indistinguishability is unncessary. In particular, it seems that hardness of $(f \Rightarrow_{\$} f)$ may be sufficient. If we further want to Φ to support transport of existing keys or actual message content, then we need anybody to find preimages of f: that is, given any y, there is an efficient algorithm to find x such that $f(x) = y$. Furthermore, the distribution of such x must be such that the hard problems defined above retain their hardness. We call this property invertibility. In this case, we also a want the stronger security property of IND-CCA2-Φ.

A similar treatment may be made for a digital signature scheme, say Σ. Here we find that the hard main problem (generating a signature on a given message) need not be restricted to a function evaluation problem, since probabilistic signatures are considered acceptable. Nevertheless, for simplicity, we assume that signatures are deterministic, so that the problem of generating a signature on a given message is an evaluation problem, say for a function f. Before getting to security properties, we first consider utility properties. As with Φ, the function f must have a

trapdoor. Unlike Φ, however, the decision problem $\mathsf{DE}-f$ must be easy, because verifying a signature amounts to solving the decision problem.

We now formally define problems EX-f and UI-f, that we have already considered for Φ. The problem UI-f, is the universal inversion problem for f: given any y, find x such that $f(x) = y$. For utility of Φ, this problem must be easy, but for security of Σ, this problem must be hard. The problem EX-f is to find (x, y) such that $f(x) = y$. This is a potentially easier problem than UI-f. Therefore, it must be easy for the utility of Φ, and this was also noted earlier. For the security of Σ, hardness of EX-f represents a stronger property than hardess of UI-f, but it too is usually considered a required property.

We remark that there seems to be something almost duality between Φ and Σ; between utility and security; between easy and hard; between the problems UI-f and DE-f; and between EX-f and CDE-f. The *almostness* is that easiness of CDE-f does not seem necessary for the utility of signatures.

We also expect Σ to resist active attacks. For example, we expect the following problem to be hard:

$$(f \Rightarrow_{\sim} \mathsf{EX}\text{-}f), \tag{25}$$

where $\sim$ indicates that the reduction is regarded to have failed if it output a solution (x, y) where x was the input to its one of its f-oracle queries.

We see that for both Σ and Φ, the usual expectations include difficulty of the problem

$$(f \Rightarrow_{\$} f). \tag{26}$$

Indeed, for both schenes, hardness of this problem is a weaker condition than full security. We now take a moment to focus on this condition, because of its simplicity and commonality to both types of schemes.

If one believes that f is hard to evaluate, one might want to infer that (26) is also hard, by find a reduction of the form:

$$(f \Rightarrow_{\$} f) \Rightarrow f \tag{27}$$

But here we find an irreduction that shows this to be unlikely.

Theorem 4. *There exists a metareduction for* $((f \Rightarrow_{\$} f) \Rightarrow f) \Rightarrow f$. *That is,* $(f \Rightarrow_{\$} f) \not\Rightarrow_{!} f$.

Proof. We write A for the desired metareduction, B for its oracle for reduction (27) and C for the oracle that B has for the reduction $f \Rightarrow_{\$} f$, and finally D for oracle that C has for f. We first note that if C were just a reduction, and not a robust reduction, then our task would be trivial. We will see why this is in detail. A adopts two strategies A_1 and A_2.

Strategy A_1 is for A to invoke B with its own input x. The strategy succeeds if B makes no C queries. Otherwise it fails.

Strategy A_2 is for A to invoke B with a random input z. When B makes a C query, say with input y, then A simulates C. As C, it makes a D query with input of x. Because x and z are independent, the fact that B can make its D oracle depend on the input y, does not affect the action of the D oracle. Note that this simultation of C will be partial in the sense A will not compute $f(y)$ for B.

Now we note that if C is not robust, then A could just use its own input x when it makes its D-query, and B would be obliged to answer it, and thus the A_1 and A_2 strategies could be merged. But C is robust, so two distinct strategies are needed.

Let β_1 be the probability that B is successful without making C queries. Then A_1 is successful with probability β_1.

Let β_2 be the probability that B is successful and that it does make some C queries. Note that $\beta_1 + \beta_2$ is the total success rate for B. Let q be the number of queries that B makes C. We suppose that C expects its oracle D to succeed with probability negligibly close to 1. Then A_2 is successful with probability at least β_2/q. The factor $1/q$ is the probability that of the q C-oracle simulations that A selects make the $D(x)$ query, that this is also the first C-oracle from B expects its answer $C(y)$. Since B has no right to expect an answer from C until it provides an answer as D, then in this case A has its answer $D(x) = f(x)$. With probability $1 - 1/q$, however, B may ask A to evaluate some $f(y)$ before A has its answer for $f(x)$.

If A selects the two strategies with equal probability then its rate of success will be $(\beta_1 + \beta_2/q)/2$. □

This irreduction is loose, but it is optimal. It is interesting in that it says a rather basic but active security property of both public key encryption and digital signatures cannot be proven, via a direct reduction, to be as strong as the corresponding basic passive security property.

H On Notations for Reductions and Oracle Assisted Problems

Perhaps the main contribution of these appendices is a unified notation for reductions between problems, and oracle assisted problems. The notations of $P \geq Q$ and $P \Rightarrow Q$ have many precedent for reductions. For oracle assisted problems, which have denoted by wrapping the reduction in parentheses as $(P \Rightarrow Q)$, there is an existing common notation, namely Q^P, such as recently used by Lipmaa [6].

Here we have used parentheses to unify the notions and notations for reductions and oracle assisted problems. This unification is orthogonal to the notation of $\Rightarrow$ or $\geq$ or the exponentiation (that is, superscripting) notation.

Assuming a unified notation and parenthesization, we can then compare the notations.

- The $\geq$ notation is intuitive because of transitivity. If $P \geq Q$ and $Q \geq R$, then $P \geq R$. However, not only can this transitivity be proved, it can be proved via a reduction. Using parentheses, one of writing this reduction is as

$$(Q \geq R) \geq ((P \geq Q) \geq (P \geq R)) \tag{28}$$

  It is not customary, and thus arguably not intuitive, to compare inequalities to each and to other problems, so we argue that $\geq$ is not the ideal notation for the calculus of metareductions. The form of transitivity above was deliberately written to look similar to the second Lukawisiewicz axiom $\beta(P, Q, R)$ given in (5). In the axiom, however, the initial predicate has an extra P oracle, which makes it a potentially easier problem. In other words, this axiom says something slightly stronger than transitivity. (This is a more nebulous reason not to use the $\geq$ notation.

- The $\Rightarrow$ notation's motivation is from logic, specifically $P \Rightarrow Q$, means that if P can be solved then Q can be solved. In formal logic, sometimes $\rightarrow$ is used instead, with $\Rightarrow$ reserved for higher level implications. However, in crypology, if not computer science more broadly,

the notation $\Rightarrow$ seems to have precedent for reductions. Indeed, here we have conjectured some informal relations between logic and the calculus of reductions. (See below for further discussion.) But we argue that this is a strong tie, and the notation $(P \Rightarrow Q)$, meaning that a P oracle is available to solve Q is easier to become accustomed to than comparison of inequalities.

- The superscript or exponent notation loses the connection to formal logic that has been discussed in this paper. It is also more difficult to attach qualifiers to the oracle access, since it does not actually use any symbol corresponding to the oracle access. Lipmaa [6] uses qualifiers in the form $Q^{P[1]}$ where the [1] indicates the number of common problems parameter that P oracle must share with the Q problem. (Here, some parameters were typically implicitly in common, while others could be related with the notations $\Rightarrow_{\sim}$.) However, the superscript has the following advantage, in that some of the rules of exponentiation also work nicely for operation. Specifically, there is a commutativity of predicate law: if $(R^Q)^P$ is easy then $(R^Q)^P$ so is. If we drop the parentheses, by introducing multiplicative notation at the superscript level, then this statement becomes: if R^{QP} then R^{PQ}. If we interpret the multiplication of problems to mean the problem solving both problems, then obviously $PQ = QP$, so multiplcation is commutative, and the law that $(R^Q)^P$ implies $(R^P)^Q$ becomes intuitive, re-using the intuition of exponentiation. Indeed, one may consider using the superscript notation more generally in logic, even standard boolean logic. Superscript notation, because it is nested, can be written without parentheses, since the parenthetic levels can be determined by the elevation of problems. Let us examine the three axioms α, β, γ of boolean logic in this notation.

$$\alpha(P,Q) = ((P^Q)^P) = P^{QP} \tag{29}$$

$$\beta(P,Q,R) = ((R^P)^{(Q^P)})^{((R^Q)^P)}) = R^{PQ^P R^{QP}} \tag{30}$$

$$\gamma(P,Q) = (P^{(P^{(Q^P)})}) = P^{P^{Q^P}} \tag{31}$$

Remebering the superscripts are oracles, one can see how α and β are easy problems, because the main problem to solve, the base problem in the exponent, appears in the exponent. Less clear is the γ problem, which we believe to be difficult for some choices of P and Q. The problem P has an oracle P in the exponent, but this P has a Q oracle. There is another P oracle higher yet, but is unclear if that helps.

One can also introduce the following extra notations. Let 0 be the problem that is impossible to solve, and the 1 be the problem that is immediate to solve. That is, all solutions to 0 are defined to be wrong, and all solutions to 1 are defined to be correct. Obviously, $1^P = 1$ and $0^P = 0$ because an oracle for P has no effect on the problems 0 and 1, while $P^1 = P$, because an oracle for 1 does nothing. We would also like to think that $P^0 = 1$, again in agreement, with the usual notations, if an oracle for the impossible problem existed, which is an impossibility, then from an impossibility anything can be derived. However, this should really be considered as undefined. Note that if we revert back to boolean logic, then 0 corresponds to false, and 1 corresponds to true. With this view, we may think of other problems as values intermediate between 0 and 1. Although the arithmetic of exponentiation carries over well, so far.

Returning back to the axioms and this notation, the fact that α is easy can be re-written

as $P^{QP} = 1$. This expression does not hold in general in, say, real arithmetic, so it gives a reason that the superscript is less than perfect, in terms of intuitivity.

The three basic notations above are qualitative. They ignore the cost of the reductions solving the problems, the number of oracle queries, and the probability of success of the reductions and oracles. These quantiative issues are very important in cryptology. Nevertheless they are not more important than the qualitative issues. It is fairly customary, at least in provable security papers, to define notations for cost of algorithms, usually with a variable t, and probabilities of success, usually with a variable ϵ. If oracle queries are involved, then they are usually indicated with a variable q. Then a definition of (t, ϵ, q)-security is given. The qualitative aspects are given in words, possibly supplement with many symbols. The notational emphasis is on the quantative aspects. That said, the qualitative aspects deserve as much notational emphasis.

So the qualtitive statement that a public-key encryption scheme, with decryption function f, secure that is secure assuming the gap Diffie-Hellman problem (where g is the Diffie-Hellman evaluation problem) may be written as,

$$(f \Rightarrow_{\sim} \mathsf{CDE}\text{-}f) \Rightarrow (\mathsf{DE}\text{-}g \Rightarrow g). \tag{32}$$

Any common parameters of the problems, such as the underlying group can be stated externally to this notation. Then quantitative parameters, can be stated, either externally, or by attachment to some of the arrows.

Obviously without quantifiers there exists reductions $P \Rightarrow Q$, namely the algorithm that solves Q. A common view in cryptology is to regard problems to be easy if they can be solved with a polynomial time algorithm. Then $P \Rightarrow Q$ can be taken to mean that the reduction is polynomial time. We do not wish to limit ourselves so strictly.

- If Q is believed to be exponential time, then a reduction $P \Rightarrow Q$, that is subexponential, is noteworthy. Even an exponential reduction is noteworthy if faster than the best known algorithms for solving Q. Therefore, an easy-hard paradigm, while useful in at the problem and security level, is much less useful at the reduction level. Setting security levels cannot be done using only the polynomial time criterion. Indeed, time is finite, and over finite intervals any continuous function may be approximated uniformly by a polynomial.

- One may think that any strict easy-hard divide may force reductions into behaving as boolean operations, because it artificially enfoces a law of the excluded middle. This would suggest Peirce's law holds at this level of detail, suggesting that Peirce's law should always be polynomial time, or even linear time. This does not appear to be true. If either problem P or Q is easy, then Peirce's law, in the form of a problem is easy, in the same sense of easy. But if both problems are hard (e.g. non-polynomial time), then one cannot conclude that Peirce's law is easy a problem. Indeed, one cannot assume that $P \Rightarrow Q$, though in boolean logic, if P and Q have the same truth value then $P \Rightarrow Q$ is true, which we are tempted correpsonding to easy. In other words, the complexity of the problem $P \Rightarrow Q$ is not merely a function of the complexities of the problems P and Q.