One-Up Problem for (EC)DSA

Daniel R. L. Brown*

June 25, 2008

Abstract

The one-up problem is defined for any group and a function from the group to integers. An instance of problem consists of two points in the group. A solution consists of another point satisfying an equation involving the group and the function. The one-up problem must be hard for the security of (EC)DSA. Heuristic arguments are given for the independence of the discrete log and one-up problems. DSA and ECDSA are conjectured to be secure if the discrete log and one-up problems are hard and the hash is collision resistant.

1 Definitions

A group G with a function $f : G \to \mathbb{Z}$ forms a *convertible group* (G, f). We will call f the *conversion* function. For generality, we will use multiplicative notation for G.

Definition 1. *The* ***one-up*** *problem for* (G, f) *is: given random* $(a, b) \in G^2$ *find* $c \in G$ *such that* $c = ab^{f(c)}$.

The tentative name *one-up* is motivated by two observations: first, the goal equation may be written as $c = a^1 b^{f(c)}$, and, second, the equation has one copy of c appearing up in the exponent, which seems to be the problem's crux. Motivation for considering this problem is:

- If certain one-up problems are easy then DSA or ECDSA signatures can be forged.
- Heuristic arguments suggest that the one-up problem is independent of the discrete logarithm.
- DSA and ECDSA appear secure if the discrete log and one-up problems are hard, and the hash is collision resistant.

Definition 2. *Let* (G, f) *be a convertible group. Let* $g \in G$. *Let* n *be the order of* g. *Require that* n *is prime. Let* $h : \{0, 1\}^* \to \mathbb{Z}$, *which is called a* ***hash*** *function on bit strings. Then* ***Convertible DSA*** *for parameters* (G, f, g, h) *is defined as follows.*

1. *A signer chooses a random integer* x *and compute public key* $y = g^x$.
2. *For message* $m \in \{0, 1\}^*$, *a signer computes a signature* $(r, s) \in \mathbb{Z}^2$ *on* m *as follows.*
   (a) *Choose a random integer* t, *such that* $\gcd(n, t) = 1$.

*Certicom Research

(b) *Compute* $r = f(g^t)$.

(c) *Compute* $s = (h(m) + rx)/t \bmod n$.

3. *Given message signer's public key* $y \in G$*, message* $m \in \{0,1\}^*$ *and purported signature* $(r, s) \in \mathbb{Z}^2$ *on this message, a verifier accepts the signature only if* $\gcd(s, n) = 1$ *and*

$$r = f\left(\left(g^{h(m)}y^r\right)^{1/s \bmod n}\right). \tag{1}$$

The signature schemes DSA and ECDSA are special cases of Convertible DSA in which the convertible group and the hash function and specified.

2 Previous Work

Somebody, no doubt, has considered the one-up problem before, just not to my knowledge in print. Obviously, to some degree it is implicit in the definition of the DSA and ECDSA. Indeed, the fact that DSA and ECDSA remain essentially unbroken testifies to the difficulty of the one-up problem. It is the aim of this work not to study the hardness of the one-up problem, but rather to isolate its role in contributing to the security of DSA and ECDSA. Several papers have addressed the security of DSA and ECDSA, but do not seem to refer to the one-up problem. Therefore, although probably discovered previously, its relevance and importance has not been well known, or at least well enough known.

In [1, 3] the *semi-logarithm* problem (SLP) and its impact on DSA and ECDSA was discussed. The semi-logarithm must be hard for the security of (EC)DSA defined with the same group and conversion function. The discrete logarithm problem must be hard for the semi-logarithm problem to be hard. What is novel here, is that the one-up problem must also be hard for the semi-logarithm. We conjecture that the converse holds. That is, if the discrete log and one-up problems are hard for a given convertible group, then so is the semi-log problem, and furthermore that the appropriate versions of DSA and ECDSA are secure.

Cao [4] introduced a problem, in the context of DSA, which may generalized as follows: given $(a, z) \in G \times \mathbb{Z}$ find $t \in Z$ such that $z = f(a^t)$. Thus, in common with the one-up problem, this problem is defined for a convertible group. Also in common with the one-up problem is that Cao's problem must be hard for convertible DSA to be secure. But there are significant differences. In cases where f is almost invertible (such as ECDSA) this problem is essentially equivalent to the discrete logarithm problem. In cases where f is almost injective (which one would conjecture to be true for DSA), then an oracle to solve this problem can be used to to solve the discrete logarithm problem. Therefore, Cao's problem is more closely related to the discrete logarithm problem than the one-up problem is. Indeed, if there is a significant gap between Cao's problem and the discrete log problem, then the one-up problem is likely to be easy, because the conversion function itself would have a security problem is isolation.

Paillier and Vergnaud [5] find a positive and a negative results about the security of DSA and ECDSA. Their positive result is that a signing oracle cannot be used to attack the private key unless a certain problem is hard, a variant of the one-more discrete logarithm problem. The fact that they did not find a positive result relating the difficulty of forgery to a variant of the discrete logarithm could possibly be explained by the independence of the one-up problem from the discrete log problem. Their negative result is a metareduction that shows that a certain type of reduction, that could be used as security proof of DSA and ECDSA, is infeasible. This hypothesized reduction would show that breaking DSA or ECDSA could be used to solve the discrete log problem. Again, this may also be attributed to the fact that the one-up problem is independent of the discrete log problem. That is, no such reduction exists, because the one-up problem could be easy without the discrete log problem being easy. (Their similar negative result about Schnorr signatures could not be explained by the one-up problem, however.)

3 Necessity

Theorem 1. *Suppose algorithm U exists with the property that $c = U(a, b)$ is a solution the (a, b) instance of the (G, f) one-up problem. Then U can be used to forge a Convertible DSA signature with parameters (G, f, g, h). More precisely, it is a universal forgery using a key-only attack (the most severe type of forgery).*

Proof. We construct an algorithm E that on input $(y, m) \in G \times \{0, 1\}^*$ will output a a pair $(r, s) \in \mathbb{Z}^2$ that a verifier will accept:

1. Choose any integer s such that $\gcd(s, n) = 1$.
2. Compute $a = g^{h(m)/s \bmod n}$.
3. Compute $b = y^{1/s \bmod n}$.
4. Compute $c = U(a, b)$.
5. Compute $r = f(c)$.

It is routine to verify that that (r, s) satisfies the verification equation (1). $\square$

4 Sufficiency?

Conjecture 1. *Convertible DSA for parameters (G, f, g, h) is secure against universal forgery against key-only attacks if:*

- *The one-up problem for (G, f) is hard,*
- *The discrete log problem for G is hard, and*
- *The hash function h is zero (modulo n) negligibly often for random messages.*

If the h meets the further necessary conditions from [1, 2, 3], then the signatures are secure against more general forgery.

Implicit in this conjecture is a secure random number generators for the signer and lack of side channel information for the adversary.

This is intended to be a rather strong conjecture, in that no specific convertible group is given. So, a skeptic may attempt to contrive pathological (G, f) which permits an attack on Convertible DSA but does not appear to admit an easy discrete log or one-up problem. Such a counterexample, if found, may further our understanding the security of DSA and ECDSA. Encouraging the search such underlying bases for the security of DSA and ECDSA is not only motivation for this conjecture, though: it is supported by some some heuristic arguments.

5 Complexity

Conjecture 2. *For the convertible groups (G, f) used in the standardized versions of DSA and ECDSA, solving the one-up problem takes about $n/2$ group operations and applications of f, to solve with reasonable probability, such as $\frac{1}{2}$ (if a solution to the instance exists).*

The conjecture says that the best algorithm for these convertible groups to be the solve the one-up problem is to exhaustively search values of c. This takes about $n/2$ operations.

The discrete log problem takes about $\sqrt{n}$ group operations. If Conjectures 1 and 2 are true, then the discrete log problem is the main bottleneck to security Convertible DSA, aside from the hash function. Attacking Convertible DSA through one-up problem is much less effective than through the discrete logarithm.

To use Theorem 1, one instance of one-up problem must be solved for each signature forged. Solving the discrete log problem (to find the signer's private key from the signer's public key) allows multiple forgeries. Therefore the one-up problem is even less useful as an attack route.

This section continues by informally comparing one-up problem (OUP) in (G, f) and to the discrete log problem (DLP) in G and other problems. Heuristic arguments are given that the OUP is independent of the DLP.

5.1 Hard DLP, Easy OUP

The OUP becomes easy if f is sufficiently degenerate. For example, if f is constant, then to $c = ab^{f(a)}$ is a valid solution since $f(c) = f(a)$. One can show that a necessary condition for the hardness of the OUP is that f is *almost injective*: no integer has an excessively large preimage under f. That this almost injectivity of f is necessary for the security of Convertible DSA has already been noted in [1, 2, 3].

We argue, however, that even if f is almost injective, then the OUP can theoretically easy even if the discrete log problem is hard. To argue, this we suppose that G is a group of prime order n. We further suppose that G is a generic group, meaning that its group operations can only be accessed via an oracle. The oracle chooses random representation for the group elements.

It is a well known result of Nechaev, and later Shoup, that the DLP in a generic group takes about $\sqrt{n}$ oracle queries to solve (with probability above, say, $\frac{1}{2}$). We argue that if the adversary attempting to solve the DLP is given access to an OUP oracle, that the DLP remains as hard.

A very similar result was already proven [2, Lemma 1], with the hint command play the role of an OUP oracle. The hint oracle and OUP oracle are not exactly the same, though, but the argument seem to go through fine.

5.2 Easy DLP, Hard OUP

Suppose that n is prime and that $G = \mathbb{Z}/(n)$. Here we will use additive notation (but retain the terminology "logarithm" used for multiplicative notation). That is, G is the group of integers under addition modulo n. Choosing the base 1, the discrete logarithm problem in G is trivial, the logarithm of x is x. Choose another base b, the discrete logarithm of x is $x/b \bmod n$, which can be computed using the extended Euclidean algorithm, for example, which is quite efficient. Discrete logs can be computed in $\log(n)^3$ steps, which is faster than $\sqrt{n}$ steps that one usually expects. We further suppose that n is large enough to make doing n steps hard. For example, $n > 2^{80}$ is pretty hard.

We suppose that f is modelled by a random oracle. Then for any c whatsoever, $f(c)$ is a random integer. For a given $(a, b) \in G^2$ instance of the (G, f) problem, and the probability that any candidate solution $c \in G$ is correct is exactly $1/n$. If an algorithm attempting to solve this OUP instance makes q queries to the random oracle for f, its probability of finding a solution is at most $(q + 1)/n$. This supports our conjecture that the complexity of OUP can be up to n steps.

Of course, for specific choices of f, it is not reasonable to model f as a random oracle. Nevertheless, if we are willing to model the group as generic, then its elements already have a random representation, and if f is efficiently computable, then we may regard f as a random oracle defined on the group $G = \mathbb{Z}/(n)$.

5.3 Semi-Logarithm Problem

Definition 3. *Let (G, f) be a convertible group. The semi-logarithm problem for (G, f) is: given random $(g, z) \in G^2$, find $(r, s) \in \mathbb{Z}^2$ such that*

$$r = f\left((gz^r)^{1/s \bmod n}\right) \tag{2}$$

In other words, a semi-logarithm would be a Convertible DSA signature of a hypothetical message whose hash is 1 (such a message need not even exist for some hashes). The following results about the SLP were given in [1, 3].

- The hardness of the semi-logarithm problem (SLP) is necessary for the security of Convertible DSA, specifically to resist universal forgery via key-only attacks.
- Hardness of the SLP is equivalent to the security of Convertible DSA against universal forgery using key-only attacks, provided that the hash function is rarely zero (modulo n).
- In the random oracle model for the hash function h, the hardness of the SLP is sufficient for security of Convertible DSA, against existential forgery using adaptive chosen message attacks.

Generally, each (g, z) may have many semi-logarithm (r, s), loosely because there are two unknowns and one equation. Similarly, roughly, for each value of one of the variables, we expect there to be an average of number solutions equal to one for the other value.

Theorem 2. *The problem of solving the SLP with a fixed pre-determined r such that $\gcd(r, n) = 1$ is equivalent to the DLP, if f is almost invertible (defined in [2]).*

Proof. If one can solve the DLP, then one can find s by finding the discrete log t of gz^r to the base u for some $u \in f^{-1}(r)$, finding this pre-image by using the almost-invertibility of f, and then setting $s = 1/t \bmod n$.

If one can solve the r-fixed SLP, then one can solve an instance of the DLP, say (h, y), whose solutions is an x such that $y = h^x$, as follows. Arrange that $gz^r = h$, by choosing $z = g^{-1}h^{1/r \bmod n}$, and that $gw^r = y$ by choosing $z = g^{-1}y^{1/r \bmod n}$. Generate instances (g, z) and (g, w) and solve the r-fixed SLP for these instances, with solutions s and t. By the almost-injectivity of f (implied by the almost invertibility), there is a good chance that $h^{1/s \bmod n}$ and $y^{1/t \bmod n}$ both map to the same r preimage $u \in f^{-1}(r)$, because the preimage cannot be too large. In that case $x = t/s \bmod n$. □

Theorem 3. *The problem of solving the SLP with a fixed pre-determined s such that $\gcd(s, n) = 1$ is equivalent to the OUP, if f is almost-invertible.*

Proof. If one can solve the OUP, then, given instance (g, z) of the SLP, let $t = 1/s \bmod n$, let $(a, b) = (g^t, z^t)$, and solve the OUP instance (a, b). On getting solution c, let $r = f(c)$, and this is a solution to the SLP instance (g, z).

If one can solve the s-fixed SLP, then, given instance (a, b) of the OUP, let $(g, z) = (a^s, b^s)$, and solve the instance (g, z) of the s-fixed SLP. On getting a solution r, choose $c \in f^{-1}(r)$. □

Based on the results above, consider the following heuristic argument. Imagine an algorithm that solves the SLP. If we look into the code of the algorithm, after sufficient reverse engineering, we may see which of r or s it computes first. If it computes r first, then the remainder of the algorithm may be viewed as an algorithm that solves the SLP with r fixed. Vice for versa for s. Thus, very loosely speaking, one expects to break one of the DLP or the OUP. This is the main heuristic argument justifying Conjecture 1.

This argument is very weak for a couple reasons. Foremost, is that it breaks the black box access rule to the algorithm for solving the SLP. Almost important is that the values r and s may be computed simultaneously, say, alternating a bit a time. More formally, if r is computed first, the algorithm may have also computed considerable state data in its memory that makes subsequent computation of s quite quick. Our heuristic argument above pretends that once the algorithm computes r it throws away all its other computations and starts over with just r. Obviously, an algorithm need not work this way, even if we could reverse engineer how it works (breaking black box access).

For lack of a better term to describe the relation between these three problems, we say that the SLP *forks* into the DLP and OUP. Some other forks are already well known, including the weak RSA problem: given an RSA modulus N and an integer y, find integers x and $e > 1$ such that $x^e \equiv y \bmod N$. The well known Strong RSA Assumption is that this problem is hard. This problem forks into the RSA problem by fixing e and into the DLP (over the RSA group) by fixing x.

From one perspective, the results on the SLP suggest that its study is more crucial for the security of Convertible DSA than the study of the DLP and OUP. That the SLP has been ignored, at least compared to the DLP may due to three things. First, its definition includes the conversion function, making it a messy problem mathematically. Second, its definition is so similar to Convertible DSA itself, making its assumed hardness seem a circular assumption. (A major non-circularity is the exclusion of the hash function from the SLP definition. Nevertheless the SLP is essentially equivalent to breaking the basic security of Convertible DSA.) Third, the SLP is a weakening of the DLP, and some may view the gap as uninteresting and not heed it attention.

The one-up problem shares with the SLP its first defect: it depends on the awkward conversion function. As such, the OUP may also defy a clean understanding. In the second aspect, however, the OUP seems less closely related to the definition of Convertible DSA. Indeed, one may be break DSA without solving the OUP at all. In the third aspect, the OUP appears more independent of the DLP. Indeed, in some sense, the OUP appears to capture the gap between the SLP and DLP. Therefore, isolating the OUP should help to serve to motivate a focus at least some further study on the security of DSA and ECDSA to the extent that it rely not merely on the discrete log problem.

5.4 Diffie-Hellman Problem

The classic Diffie-Hellman problem (DHP) is: given g, and g^x and g^y, compute g^{xy}, and we write $g^{xy} = g^x \otimes_g g^y$ or just $g^x \otimes g^y$ when g is clear from context. Let g have order n. Here we consider the following variant: given g, g^x and g^y, determine $g^{x/y \bmod n}$, if $\gcd(y, n) = 1$. We write $g^{x/y} = g^x \oslash_g g^y$ or $g^x \oslash g^y$ when g is clear from context. We call this variant problem the inverted Diffie-Hellman problem (IDHP).

An algorithm to solve the IDHP can be used to solve the DHP, if $\gcd(y, n) = 1$, because $g^x \otimes g^y = g^x \oslash (g \oslash g^y)$. If n is known, which it typically is, then an algorithm to solve the DHP can be used to solve the IDHP, because $g^x \oslash g^y = g^x \otimes g^{y^{n-1}}$ and $g^{y^{n-1}}$ can be compute quite efficiently using the operation $\otimes$ with a square-and-multiply algorithm acting on the exponent of g.

Now suppose that G is generated by g and the conversion function is the discrete log function to the base g. So $f(g^x) = x$. Then the one-up problem on input (a, b) is to find a c such that $c = ab^{f(c)} = a(b \otimes c)$. Then $c = a^{-1} \oslash bg^{-1}$. Therefore the if the IDHP (and hence DHP if n is known) is easy, then the OUP for this (G, f) is easy. Conversely, if $(a, b) = (d^{-1}, eg)$, and c is an OUP solution for instance (a, b), then $c = d \oslash e$. Therefore, the IDHP is easy if the OUP is easy. In other words, for this choice of f, the OUP is equivalent to the IDHP, and if n is known, which is typical, then this OUP is equivalent to the DHP.

Unfortunately, for the purposes of Convertible DSA we are mainly interested in the OUP for cases f is efficiently computable, which excludes this example. It does not seem likely that this special case of the OUP is related to the DHP.

6 Conclusion

The one-up problem was identified. The version DSA and ECDSA must be hard for these signature schemes to be secure. Heuristic arguments suggest the following.

- The one-up problem generally has complexity n, where n is the order of the group.
- The one-up problem and discrete log problems are generally independent.
- DSA and ECDSA are secure if their associated one-up and discrete log problems are hard, and their associated hash is secure.

References

[1] D. R. L. Brown. A conditional security analysis of the elliptic curve digital signature algorithm. In *Elliptic Curve Cryptography 2001*. Presentation at `http://www.cacr.math.uwaterloo.ca/conferences/2001/ecc/brown.ps`.

[2] D. R. L. Brown. Generic groups, collision resistance, and ECDSA. *Designs, Codes and Cryptography*, 35:119–152, 2005. `http://eprint.iacr.org/2002/026`.

[3] D. R. L. Brown. On the provable security of ECDSA. In I. F. Blake, G. Seroussi, and N. P. Smart, editors, *Advances in Elliptic Curve Cryptography*, volume 317 of *London Mathematical Society Lecture Note Series*, pages 21–40. Cambridge University Press, 2005.

[4] Z. Cao. On the big gap between $|p|$ and $|q|$ in DSA. ePrint 2007/320, IACR, 2007.

[5] 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 In or up?

Schnorr signatures may be generalized similarly to Convertible DSA. Now let $i : \mathbb{Z} \to \{0,1\}^*$. Let $\|$ indicate concatenation of bit strings. Convertible Schnorr signatures for parameters (G, f, g, h, i) have the same key pair generation as Convertible as Convertible DSA, with signing algorithm

1. Choose a random integer t.
2. Compute $r = f(g^t)$.
3. Compute $e = h(m\|i(r))$.
4. Compute $s = t + ex \bmod n$.
5. The signature is $(e, s) \in \{0,1\}^* \times \mathbb{Z}$.

Verify a signature (e, s) on message m under public key y only if

$$e = h\left(m\|i\left(f\left(g^s y^{-e}\right)\right)\right) \tag{3}$$

A variant of this, which highlights the similarity to Convertible DSA, is to make the signature (r, s), and to verify via

$$r = f\left(g^s y^{-h(m\|i(r))}\right) \tag{4}$$

This variant is equivalent in the sense that given m and the valid signature of one type, anybody can easily produce a valid signature of the other type. (On the other hand, the form (e, s) may have the advantage of having smaller data size, depending on the parameters (G, f, g, h, i).)

In Schnorr signatures, the value r is included in the hash function argument. To see that this is important, consider what would happen if it were omitted. The verification equation would then be

$$r = f\left(g^s y^{-h(m)}\right) \tag{5}$$

With no r on the right hands side, the equation can be used to compute r from public information and arbitrary s. Universal forgery using key-only attack is possible. Note that the verification equation for Convertible DSA has r on the right hand side. A major difference between Convertible DSA and Convertible Schnorr is that the former puts it *up* in the exponent directly and latter puts it *in* the hash argument (albeit also up, indirectly).

A natural and frequently asked question is which location for r on the right side is superior. In or up? Proofs of security due to Pointcheval and Stern in the random oracle model (for the hash function) suggest that *in* is better because it can be shown the hardness discrete log problem suffices in this model. Therefore, with *in* there is little fear of some problem such as the one-up problem lurking behind the security. Indeed this fear has to a degree borne out in the case of Convertible DSA, except that the undiscovered problem is actually harder than the discrete log problem.

On the other hand, the security of Convertible DSA against universal forgery using key-only attacks has been proven to depend only on the hardness of the semi-logarithm problem and the hash being rarely zero [3, 1]. This is an absolutely essential security property for a signature scheme, and Convertible DSA barely relies at all on the hash to achieve it. Convertible Schnorr, when not relying on a random oracle, seems to have this fundamental security property hinge crucially on the hash function.

In and up is what one can call another suggestion, first made by Pointcheval and Stern: to modify DSA and ECDSA to also include r in the hash arguments, as done in Schnorr. The modified verification equation is then

$$r = f\left(\left(g^{h(m\|i(r))}y^r\right)^{1/s \bmod n}\right). \tag{6}$$

Note that r appears twice in the exponent, so this approach may also be called *doubled up*. Conventional thinking is that this combines the best of both Schnorr and DSA. Note however, that like Schnorr signatures, it more intricately ties the role of the hash function in the most essential security property: universal forgery against key only attacks. Another common concern with DSA and ECDSA is that a collision in the hash leads to an active attack. While true, for iterated hash function such as SHA-1, it is typical that if $h(m) = h(m')$, then $h(m\|r) = h(m'\|r)$, so collisions still lead the same attack on Schnorr and doubled up DSA. If r is prepend to the message, then perhaps this is avoided (although some of the collision attacks on MD5 are close to handling arbitrary IV), but then the some practicality is lost: one cannot precompute hash values of messages.