Short Schnorr signatures require a hash function with more than just random-prefix resistance

Daniel R. L. Brown[*]

February 27, 2015

Abstract

Neven, Smart and Warinschi (NSW) proved, in the generic group model, that full-length Schnorr signatures require only random-prefix resistant hash functions to resist passive existential forgery.

Short Schnorr signatures halve the length of the hash function, and have been conjectured to provide a similar level of security. The NSW result is too loose to provide a meaningful security for short Schnorr signatures, but Neven, Smart and Warinschi conjecture that this is mere artefact of the proof technique, and not an essential deficiency of the short Schnorr signatures. In particular, this amounts to a conjecture that short Schnorr signature are secure under the same set of assumptions, namely random-prefix resistance of the hash function.

This report provides a counterexample to the latter conjecture, in other words, a separation result. It finds a hash function that seems to suggest random-prefix resistance does not suffice for short Schnorr signatures. In other words, the loose reduction implicit in the NSW theorem is as tight as possible.

Obviously, this result does not preclude the possibility of another proof for short Schnorr signatures, based on different hash function security properties such as preimage resistance.

1 Introduction

1.1 Background and Motivation

Neven, Smart and Warinschi [NSW09] proved the security of Schnorr signatures in the generic group model assuming some standard model assumptions

[*]Certicom/BlackBerry, dbrown@certicom.com

for the hash function used in the signature. The result they prove requires that the hash function width to be twice the security level. Nonetheless, they conjecture that the hash function really only needs to be half this length, provided that the hash can resist the Kelsey–Kohno herding attack. In particular, they suggest using a wide-pipe hash design, by truncating a 256-bit wide hash to 128 bits, when using Schnorr at the 128-bit security.

Neven, Smart and Warinschi mention the Kelsey–Kohno herding attack, which shows that a 128-bit narrow-pipe Merkle–Damgard hash using with Schnorr would only provide 96 bits of security. But this hash fails to meet their requirement of providing random-prefix preimage resistance at the 128-bit security level.

1.2 Other Related Work

Numerous papers prove things about the security of Schnorr signatures. Others even find limits the provable security, that is, separation results, that work by finding meta-reductions, reductions about reductions. I have not reviewed all these papers.

A prudent reader might presume that one these earlier papers contains a general result that subsumes the result of this report. When I find which one that is, I will update this report accordingly.

For now, I wrote this report as if the Neven–Smart–Warinschi eprint were the latest word on provable security and Schnorr signatures.

1.3 Contribution of this Report

This report describes a hash function that seems to provides 128-bit RPP resistance, yet, when used in Schnorr signature only provides about 67-bit security. Specifically, when instantiating the Schnorr signature scheme at the 128-bit security level, using this report's counterexample hash function, passive existential forgery becomes feasible at a cost of about 2^{67} group operations and about 2^{67} bits of memory, but success rate near to one. The combined cost and success rate exceeds what one expects for 128-bit security.

Our counterexample hash can be viewed as a variant of Bellare–Micciancio construction called AdHash. This will be defined precisely. The attack will be described. An argument of the RPP (and tentatively RPSP) security the hash function will be given.

Just to be clear: the counterexample does not contradict the Neven–Smart–Warinschi theorem, because although the counterexample hash appears to provide about 128-bit security of (chosen-target) random-prefix preimage (RPP) resistance, the NSW theorem provides a loose reduction from which one can only conclude that the corresponding instantiation of Schnorr signatures provides half the hash's RPP security level, so 64 bits in this case.

So, the counterexample hash suggests that, to prove short Schnorr signatures fully secure, requires some additional security properties from the hash function, beyond the RPP (and RPSP) resistance addressed in Neven–Smart–Warinschi analysis.

2 Definitions

The notation here loosely follows the Neven–Smart–Warinschi, though is not as precise. In particular, we use multiplicative notation, even though we are mainly concerned with elliptic curve groups nowadays.

2.1 Schnorr Signatures

We say a pair (s, h) of integer is a *valid Schnorr signature* of bit string message m under public key y (a group element), if

$$h = H(f(g^s y^{-h}) \| m) \tag{1}$$

where g is a fixed element of the same group as y, and H is hash function mapping bit strings to integers, and f is a function mapping group elements to bit strings.

We may occasionally refer to arbitrary pairs (s, h) in the same form as above as *Schnorr signatures* even if they are invalid with respect to a given public key.

The size of group used is q, which is usually prime. For the sake of this report assume that $q \approx 2^{256}$.

We further assuming that H outputs integers between 0 and $2^n - 1$ for some n. If $n = 256$, that is $q \approx 2^n$, then we are using *full-length* Schnorr signatures. If $n = 128$, that $2^n \approx \sqrt{q}$, then we are using *short* Schnorr signatures.

This report focuses on short Schnorr signatures.

2.2 BadHash

Let H_1 and H_2 be two distinct 128-bit hash functions (again with integers values as outputs). We can think of these as random oracles for our purposes, but in practice these functions could be instantiated with a practical conventional hash function like SHA-256, fixed prefixes, and truncation.

We define our “BadHash” function as:

$$H(R\|m) = H_1(R) + H_2(m) \tag{2}$$

where + can be any efficiently invertible binary operation. To be concrete, we may suppose that + is addition modulo 2^{128}.

We may sometimes further consider H_2 to be injective. This restricts the message m to be a bit string of at most 128 bits. In this case, we will call the function WorseHash.

Please note: I do not know of a practical way to instantiate WorseHash. In other words, WorseHash will be used for something even weaker than just a condtional attack: an imaginary conditional attack.

3 Random-Prefix Preimage Resistance

In this section, we attempt to argue how BadHash provides (chosen-target) random-prefix preimage resistance.

If BadHash failed to be RPP resistant, then BadHash would just be another example of the RPP weak hash, much like the single-width pipe Merkle–Damgard hash. In this case, BadHash would just seem to affirm that NSW suggestion that the looseness of their proof is indeed an artificial figment of the proof method.

If BadHash is RPP resistant, then BadHash shows that the loose reduction in the relevant corollary of the NSW theorem (or proof) cannot be improved to a tight reduction. In this case, BadHash seems to contradict the NSW suggestion that the looseness of their proof is a mere artificial figment of the proof method.

Aside: I speculate that this looseness of the NSW proof is not a defect of the proof method, but rather due too weak assumptions about the hash function. Indeed, if BadHash can be RPP resistant, then the hash assumptions are too weak. So, what I am speculating is that a tight reduction for short Schnorr signatures can be found using some other assumptions about the hash function.

So, to bolster the significance of BadHash to Schnorr signatures and its provable security, this section tries to argue that BadHash is RPP resistant.

We also argue that WorseHash seems to provide RPSP resistance too.

3.1 Review of Hash Attack Definitions

This section reviews the two NSW definitions, and one other definition, for attacks on hash functions.

3.1.1 Random-Prefix Preimage

A (chosen-target) random-prefix preimage (RPP) attacker can be viewed as pair of algorithms (A_1, A_2), with the following properties.

The first algorithm A_1 has no input and outputs a chosen target h. (The NSW also outputs a state value. We ignore this, here, but one can imagine that it is a fixed value for the purposes of this report.)

The second algorithm A_2 receives h and also an input bit string R and outputs a bit string message m.

The RPP attacker wins if $h = H(R\|m)$.

In other words, if we fix the random tapes in a run of (A_1, A_2) then the success condition is $A_1() = H(R\|A_2(A_1(), R))$. More precisely, the two runs of A_1 in the success condition are identical.

3.1.2 RPSP

An RPSP attacker can be viewed as a pair of algorithms $A = (A_1, A_2)$.

The first algorithm A_1 has no inputs and outputs a chosen message m. (The NSW definition also outputs a state, which will be considered a fixed value of the purposes of this report.)

The second algorithm is given a m and a random R and outputs message m'.

The RPSP attack wins if $H(R\|m) = H(R\|m')$.

3.1.3 Preimage Finder

A (random-target) preimage finder F, takes as input input and random candidate hash value h and outputs message m. It succeeds when $h = H(m)$.

3.2 Some Attacks Against BadHash

This subsection lists some attacks against the BadHash that I have been able to come up with. If these turn out to be the best attacks, then these characterize the security of BadHash.

3.2.1 An RPP Attack Against BadHash

My best RPP attack algorithm (A_1, A_2) against BadHash, has A_1 choose any h, either fixed, or at random. Then $A_2(h, R)$ works by exhaustively searching through values m_i until finding one such that $H_2(m) = h - H_2(R)$.

3.2.2 An RPSP Attack Against BadHash

Let C be collision finder in H_2: which finds m and m' such that $H_2(m) = H_2(m')$. Then define A_1 and A_2 output m and m' respecitvely.

Exisentially, this attacker is super efficient, but even allowing for the realistic cost of C to constructing the collision, this attack costs 2^{64} steps against an 128-bit hash function.

In the case of WorseHash, we imagine that H_2 is injective, and therefore collision-free. That seems to thwart the RPSP algorithm above.

Note that the best one-way injective function that I know would be the elliptic curve discrete logarithm. But a 128-bit size function H_2 based on this would take only about 2^{64} steps to invert. In other words, I do not know of an example of 128-bit H_2 that could achieve both 128-bit preimage resistance (need to resist the RPP attack from the previous section) and injectiveness to resist the RPSP attack above.

3.2.3 A Preimage Attack Against BadHash

The preimage resistance of BadHash is at most about 64 bits, because of the following attack.

Generate a list of of 2^{64} values m_i and compute $u_i = h - H_2(m_i)$. For practicality, sort this list according to u_i, (or perhaps store the list using some kind of hash-table, for some other non-cryptographic hash function).

Generate a list of 2^{64} values of R_j and compute $v_j = H_1(R_j)$. For each j, check if $v_j = u_i$ for some i. (If the (m_i, u_i) list is sorted, then use a binary search to find this i). By the birthday surprise effect, one expects to find a match.

Once a match $u_i = v_j$ is found, output (R_j, m_i) as a preimage of the h.

Note the this preimage attack happens to approximately match the cost of the forgery attack on short Schnorr signatures with BadHash. This suggests that adding preimage resistance to the set of assumptions might suffice to prove the security.

3.3 Informal Proofs of BadHash Security

The previous section only listed the attacks against BadHash that I was able to devise. But perhaps, I was not able to find the best attacks, and therefore the attacks of the previous may be well above the actual security level of BadHash.

Therefore, I supply further heuristic arguments for the security of BadHash and WorseHash.

3.3.1 RPP Security in the Random Oracle Model

Suppose that H_1 is a random oracle, specifically a re-programmable random oracle. Supposing this makes the following argument heuristic.

Assume that H_2 is one-way in the sense that a preimage-finder for $H_@$ is infeasible. More precisely, for 128-bit security and 128-bit wide H_2, we assume that any preimage finder F taking time t with success rate s is such that $t/s \approx 2^{128}$. In other words, the best possible preimage finder is no better than exhaustive search. This seems to be a plausible assumption for some hash functions.

We now describe a reduction which takes an RPP attacker (A_1, A_2) that operates against BadHash even when H_1 is a random oracle, and whose success rate is defined, as probability, over the random oracle choices made for H_1. The RPP attacker may be specific to a particular hash H_2. We will convert (A_1, A_2) into a preimage-finder P for H_2.

So, P receives an input h_2 whose preimage it needs to find. First P calls A_1 with no input. Obviously A_1 may query its random oracle for H_1. Algorithm P simulates a random oracle for H_1 exactly per the definition of a random oracle. At some point A_1 completes, returning a chosen target h. Next, P selects a value R uniformly at random. We note that R is negligibly likely to one the H_1 query inputs made by A_1, because it is the size of R as a bit string will be at least 256 bits.

Next P invokes the algorithm A_2 with inputs (h, R). Again, A_2 can call its random oracle for H_1. Now P_2 continues its random oracle, but adds one exception, which is that it prescribes $H_1(R) = h_1 = h - h_2$. If A_2 never calls the H_1 oracle with input R, we just let P make the call after A_2. Eventually, A_2 returns a candidate message m.

If (A_1, A_2) is successful, then $h = H(R\|m) = H_1(R) + H_2(m)$. This implies that $H_2(m) = h - h_1 = h - (h - h_2) = h_2$ as desired. So P has at least the same success rate as (A_1, A_2).

Note that the run-time of P is proportional to the total run-time of

(A_1, A_2). But then the memory used by P is also proportional to the rumtime of (A_1, A_2). We could try to reduce the memory by simulating the random oracle by a keyed pseudorandom function. Formally, speaking this would also requiring assuming some properties for the pseudorandom function. Of course, H_1 does not have a key itself, so this would still be a heuristic argument.

3.3.2 Neven's Standard Model Proof of BadHash RPP Resistance

Gregory Neven ever so kindly provided me a superior argument for the RPP resistance of BadHash. Its superiority is that it does not rely on H_1 being a random oracle. Instead it need merely be a smooth function.

As before we are trying to build a preimage finder P against the hash function H_2. Suppose that $A = (A_1, A_2)$ is an RPP adversary to BadHash. We will use A as subroutine of P.

By definition, P first receives its challenge value h_2. Now we run A_1 which outputs a value h.

Our next step is find an R value such $H_1(R) = h - h_2$. In the previous argument, we did this by using an overly powerful random oracle model for H_1. But Neven points the argument also works if H_1 is just some easily invertible function with nearly equal image sizes. For example if R has size 256 bits and H_1 has size 128 bits, then H_1 could consist of a truncation function.

Aside: essentially this property is what I called almost invertible in my ECDSA security proof paper.

In other words, the function H_1 does not really to need to be very secure at all in order for BadHash to be RPP resistant.

3.3.3 Arguments for WorseHash

If H_2 is actually injective, then no RPSP adversary against WorseHash exists, because success of a RPSP adversary implies a collision in H_2.

4 Passive Existential Forgery

This section describes the actual attack that arises when short Schnorr signatures are used with BadHash (or WorseHash).

4.1 Definining Passive Forgery

By passive forgery, I mean the forger does not make any queries to a signing oracle. Two other term for passive forgery are no-message attack and (public) key-only attack. In the notation of the NSW paper, passive forgery is when $q_S = 0$.

A forger opposite to a passive forger has access to a signing oracle, and is called an active forger, or (adaptive) chosen-message attacker. A passive forger is generally more damaging than that active forger, because real world signers can be careful about what they sign.

Some active forger can be considered more realistic and closer to this report's definition of passive forger if they have a signing oracle but have no control over the messages signed. Arguably, these forgers could be called passive too, because they are only observers. Anyway, the passive forger to be constructed

4.2 Defining Existential Forgery

By existential forgery, I mean that the forger wins even if the forged message is meaningless nonsense. In other words, no requirements are place on the forged message.

Formally, the forged message is an output of the forger, not an input to the forger. Therefore, the forger can choose whatever message it finds easiest to forger. (Note: I regularly find this formalism confusing, because it seems the forger can forge any message it wants, but actually it's the opposite.) The forger opposite to an existential forger must be able to forge any message it is given (from some large set or distribution of messages), and is called a selective or a universal forger.

4.3 The Forgery

This section describes passive and existential forger F of short Schnorr signatures when used with BadHash. For sake of concreteness, assume a 128-bit security level, with a 128-bit hash and a group size of approximately 2^{256}.

First generate 2^{64} random message m_j. Compute $v_j = H_2(m_j)$. Maintain a list of pairs (v_j, m_j), sorted (or hash-key-tabled) by value of v_j.

Second, generate 2^{64} random signatures (s_i, h_i), where s and h are integers of the appropriate sizes for short Schnorr signatures. For each signature compute:

$$u_i = h_i - H_1(f(g^{s_i} y^{-h_i})) \tag{3}$$

For each u_i check if it is in the list of v_j. If a match $u_i = v_j$ is found, then output (s_i, h_i) as the forgery of m_j under public key y.

Because the integers u_i and v_j are essentially selected randomly from sets of size 2^{128}, we expect a match to be found. Of course, this conclusion requires some assumptions about H_1, H_2 and f. In particular, they must not be too lossy as functions.

4.4 Implicit Corollary to the NSW Theorem

In the NSW theorem, put $q_S = 0$, and we can derive a corollary. The immediate corollary has a dependency on the advantage of an RPSP adversary against the hash function. To strictly contradict the theorem, it is necessary to remove this dependency from the corrollary.

It may be possible to adjust the proof, somehow removing the sligh dependency on an RPSP hash adversary only appears as a match to the generator or the public key.

Regardless when I walk through the proof, using the forger above, I find that the RPSP case can be avoided.

The quality of the result also depends on the number of generic group oracle queries made. By pre-computing a handful of powers of g and of y, we can the keep the number q_G of group queries used by our forger not too far above 2^{64}. (Maybe 2^{67}.)

This attack should be considered to faster than allowed for 128-bit security because its success rate is near to one while its runtime is not too far above 2^{64}.

Aside: this forger uses are large amount of memory to store the list of pairs (v_j, m_j). Even more convincing would be forger with a lower, more realistic, memory cost.

Anyway, short Schnorr signatures with BadHash demonstrate this immediate corollary, which is a loose reduction, cannot be tightened. The looseness in the reduction is due to a potential attack, and is not merely due to a weakness in the proof technique.

5 Conclusion and Questions

One advantage of shortened Schnorr signatures, compared to DSA and ECDSA signatures, is a smaller signature. For example, at the 128-bit security level, a short Schnorr signature is about 384 bits long, while an ECDSA signature is about 512 bits long.

The security of using short Schnorr signatures seems to be supported mainly by conjecture, not by a reductionist security argument. In particular, the Neven–Smart–Warinschi theorem only supports a full length hash in Schnorr signatures.

An observation about Schnorr signatures is that the chosen-target type attacks of the hash function seem to lead to forgeries. Chosen hash targets appear in both the RPP attack and in our forgery against short Schnorr with BadHash. Yet, ECDSA does not appear to be require chosen-target type security properties for the hash function, at least for the case of passive existential forgery. It seem as though the difference lies in the signing equation. In ECDSA, the r value is fed directly into the signing equation and not into the hash, which seems to thwart chosen-target-type hash attackers.

This result can be viewed as an instance of the slightly prescient nature of formalized provable security. Neven, Smart and Warinschi's intuition was that short Schnorr signatures would be secure provided that the hash was RPP resistant. Yet they also understood the limitations of their formal proof. In this case, the proof won out over intuition.

Again, it may well be that short Schnorr signatures are secure with a double-width pipe hash function, or other types of hash function . This report shows that any proof this conjecture will have to assume beyond RPP and RPSP resistance.

References

[NSW09] Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Hash function requirements for Schnorr signatures. *Journal of Mathematical Cryptology*, 3(1):69–87, January 2009.