Schnorr signature integration in SNARKs

Let p be the order of the prime field $F = F_q$ of the SNARK. In order to verify Schnorr signatures via SNARK, we need express scalar multiplication of elements of $\mathbb{G} = EC(F_q)$ as arithmetic circuit over F, which is a somewhat subtle issue as the group order of $\mathbb{G}$, say r, differs from the order of F, i.e.

$$r = |G| \neq |F| = q$$

In our setting, $\mathbb{G} = MNT_4(F_q)$ and both $r < q$ are of the same bit length, $L = len(r) = len(q)$.

We discuss two ways to integrate signature verification (using the Poseidon Hash PH_F over $F = F_q$) in a secure way,

1. by length restriction of the output of the Schnorr signature, or
2. by a looser definition of what verifiability means,

both of which reduce practical security by at (very roughly estimated) most 2 bits.

Finally, we explain why we favor Variant 1 for our *ginger* implementation.

Variant 1: Size-restricted Schnorr signature

System parameters, key generation and signature verification are the same as for the ordinary Schnorr signature. The only difference is that signing is repeated until both the "challenge hash" e (as Poseidon PH_F over the SNARK field F) as well as the "response" s are at least one bit smaller than the modulus r.

Primitive

Signing

$(e, s) \leftarrow Sign(m, sk, pk, pp)$: takes as input m (parsed as sequence of F-elements), the secret signing key sk and it's public verification key $pk = G^{sk}$, and outputs a pair of integers (e, s) both of bit length at most $L - 1$.

1. The signer repeats
   - drawing randomnesses $k' \overset{\$}{\leftarrow} \mathbb{Z}_r$ (uniformly from $\mathbb{Z}_r$), and computing
   - $e_F = PH_F(m, pk, R = G^{k'})$,
   - $s = (e_F \bmod r) \cdot sk + k' \bmod r$

   until both e_F and s regarded as integers from $[0, p)$ and $[0, r)$, respectively, are of bit length $\leq len(r) - 1$.
2. The output is $\sigma = (e, s)$, both of bit length at most $L - 1$.

Note: This signing process will need

$$T = \frac{p}{2^{L-1}} \cdot \frac{r}{2^{L-1}} < 4$$

attempts on average, if both p and r are of same length. For $\mathrm{MNT4\text{-}753}$ we have $T \approx 1.768^2 = 3.128$ attempts on average.

Note: there is a slightly less secure variant, which only takes the x coordinate R_x instead of R for PH_F. However, for this reason this can be more efficiently implemented in a SNARK.

Verification

Verification $1/0 \leftarrow Verify(m, pk, (e, s), pp)$ takes as input a message m, parsed as F-elements, regards (i.e. loads) e as field element e_F and simply checks if

$$e_F = PH_F(m, pk, G^s \cdot pk^{-e}).$$

Note on security

Security of the Schnorr signature is not affected by our restriction, since the challenge space (for the e's) is still large enough (in practice as well as asymptotically) to preserve the proof of knowledge property (in the random oracle model), and on the other hand since the response space (for the s's) is large enough, σ can be still simulated in probabilistic polynomial time to preserve zero-knowledgeness.

In practice it is hard to evaluate the security loss by the restriction on e and s. In regard to generic attacks it can be estimated by the proportion

$$\frac{2^{len(r)-1}}{r} < 1,$$

both for e and s, and hence about 2 bit of security, but much smaller when choosing the prime order r to be close to $2^{len(r)-1}$.

More accurate, for $\mathrm{MTN4\text{-}753}$ we have

$$\frac{2^{len(r)-1}}{r} \approx 0.565,$$

yielding 1.131 bit security loss against generic attackers.

SchnorrVerify: expressing signature verification over the SNARK field

The reason for our restriction to $L-1$ bits is that the arithmetic circuit for scalar multiplication in $\mathbb{G} = EC(F)$ (aka double and add, or square and multiply) takes as input the bits $(b_i)_{i=0}^{L-2}$ from the integer representation of the exponent b, and this relation is easily expressed over F by

$$0 = b_i \cdot (b_i - 1),$$
$$b = \sum_{i=0}^{L-2} b_i \cdot 2^i \text{ in } F,$$

since bit length $L-1$ guarantees that no modular reduction is used in the latter.

In our gadget for signature verification,

$$SchnorrVerify(m, pk, e, s)$$

we consider m as single F-element, load both e and s as F elements, allocate private F-elements $(e_i)_{i=0}^{L-2}$, $(s_i)_{i=0}^{L-2}$, private elliptic curve points R, U, V from $\mathbb{G} = EC(F)$, and enforce that

$$\begin{aligned} 0 &= e_i \cdot (e_i - 1), & 0 &= s_i \cdot (s_i - 1), & i &= 0, 1, \ldots, L-2, \\ e_F &= \sum_{i=0}^{L-2} e_i \cdot 2^i, & s_F &= \sum_{i=0}^{L-2} s_i \cdot 2^i, \end{aligned}$$

and further

$$\begin{aligned} e_F &= PH_F(m, pk, R) \\ U &= ECadd(V, R) \\ U &= SquareAndMultiply(G, (s_i)_{i=0}^{L-2}), \\ V &= SquareAndMultiply(pk, (e_i)_{i=0}^{L-2}). \end{aligned}$$

When building the proof, the prover of course calculates the private witness R as $R = U \cdot V^{-1}$.

Note: As mentioned above, if we use only R_x in the PH_F for e_F, the input of the Poseidon Hash is one field element smaller, which results in less restrictions for the Poseidon Hash verification.

Variant 2: Relaxed Signature verification

If we allow e_F and s_F in our circuit to be of full length L, i.e.

$$\begin{aligned} 0 &= e_i \cdot (e_i - 1), & 0 &= s_i \cdot (s_i - 1), & i &= 0, 1, \ldots, L-1, \\ e_F &= \sum_{i=0}^{L-1} e_i \cdot 2^i, & s_F &= \sum_{i=0}^{L-1} s_i \cdot 2^i, \end{aligned}$$

then, as integer,

$$e_F = \sum_{i=0}^{L-1} e_i \cdot 2^i - \epsilon \cdot q, \quad \text{where } \epsilon \text{ is either } 0 \text{ or } 1,$$

depending on whether the sum exceeds the modulus or not.

This means that the (e_i) are the integer bits of either e_F or $e_F + q$, and the same holds for s_F. Hence the statement of the SNARK is *"I know field elements e_F and s_F such that for one of the four combinations*

$$(e_{\epsilon_1}, s_{\epsilon_2}) = (e_F + \epsilon_1 \cdot q, s_F + \epsilon_2 \cdot q) \bmod r, \quad \varepsilon_i \in \{0, 1\},$$

satisfies

$$e_F = PH_F(m, pk, G^{s_{\epsilon_1}} \cdot pk^{-e_{\epsilon_2}}).$$

This way of talking about the verification equation forces us to

- widen the notion of signature validity, allowing all these combinations as valid, and also
- modify the signing process to one that potentially produces all these combinations.

Primitive

Signing

$(e_F, s) \leftarrow Sign(m, sk, pk, pp)$: takes as input m (parsed as sequence of F-elements), the secret signing key sk and it's public verification key $pk = sk \cdot G$, and outputs $(e_F, s) \in F \times \mathbb{Z}_r$.

1. The signer
   - draws a randomness $k' \overset{\$}{\leftarrow} \mathbb{Z}_r$ (uniformly from $\mathbb{Z}_r$), computes
   - $e_F = PH_F(m, pk, R = k' \cdot G)$, and

    - chooses either $e = e_F \bmod r$ or $e = e_F + q \bmod r$ for computing
    - $s' = e \cdot sk + k' \bmod r$.
2. It chooses either $s = s'$ or $s = s' - q \bmod r$ and outputs (e_F, s).

Notice that there is an equivalent variant which outputs (R_x, s) instead of (e_F, s).

Verification

Verification $1/0 \leftarrow Verify(m, pk, (e_F, s), pp)$ takes as input a message m, parsed as F-elements, and checks if one of the four combinations $(e_{\epsilon_1}, s_{\epsilon_2}) = (e_F + \epsilon_1 \cdot q, s + \epsilon_2 \cdot q) \bmod r$, $\epsilon_i \in \{0, 1\}$, satisfies

$$e_F = PH_F(m, pk, G^{s_{\epsilon_2}} \cdot pk^{-e_{\epsilon_1}}).$$

Note on security

With this freedom in choosing e and s, we loose just two bits of security in practice against generic attacks.

SchnorrVerify

In our gadget for signature verification,

$$SchnorrVerify(m, pk, e, s)$$

we consider m as single F-element, load both e and s as F elements, allocate private F-elements $(e_i)_{i=0}^{L-1}$, $(s_i)_{i=0}^{L-1}$, private elliptic curve points R, U, V from $\mathbb{G} = EC(F)$, and enforce that

$$\begin{aligned} 0 &= e_i \cdot (e_i - 1), & 0 &= s_i \cdot (s_i - 1), & i &= 0, 1, \ldots, L-2, \\ e_F &= \sum_{i=0}^{L-1} e_i \cdot 2^i, & s_F &= \sum_{i=0}^{L-1} s_i \cdot 2^i, \end{aligned}$$

and further

$$\begin{aligned} e_F &= PH_F(m, pk, R) \\ U &= ECadd(V, R) \\ U &= SquareAndMultiply(G, (s_i)_{i=0}^{L-1}), \\ V &= SquareAndMultiply(pk, (e_i)_{i=0}^{L-1}). \end{aligned}$$

Aftermath: why we choose length restriction

Using $\mathbb{G} = \text{MNT4-753}$ yields a performance drawback in the signing procedure by a factor of 3.128 on average, and of factor 5.532 when extending the method to the VRF proof creation. For general Schnorr-like NIZKP, we have

$$\approx 1.7686^{1+\text{number of linear conditions}}$$

times slower performance compared to the non-restricted variant.

The reasons why we still favor Variant 1 over Variant 2 are:

- Variant 2 implies a widened notion of verifiability, which is very specific to our environment. Hence one can create signatures which are verifiable by the SNARK circuit, but not per se outside our environment.

- More important, hashing Variant 2 signatures via POSEIDON demands secure packaging as field elements, which rely on expensive comparisons by value. In contrast, such comparison comes cheap for the length-restricted variant.