A short note on the (in)ability to embed data into Schnorr

1 Abstract

We demonstrate that, under already required assumptions (namely ECDLP hardness and hash function preimage resistance), the ability to embed data into a tuple (P, σ) where P is a public key and σ a BIP340 Schnorr signature on that pubkey, can only come from:

- sidechannel information
- grinding
- revelation of secret key material

Since the second item only allows the embedding of a few bytes it is ignored, here. The first example (as an example, using symmetric key encryption, or using tweaked public keys with tweaks shared with counterparties) may be important but is not considered in this document.

We thus focus on the third case; and we emphasize that **this note does not prove that data cannot be embedded, but specifically that embedding data reveals the secret witness (key)**. We use a reduction argument to demonstrate that without breaking the hash preimage resistance assumption, it is not possible to embed 256 bits of data or more into the tuple (P, σ) without at the same time revealing the secret witness. This may have implications for a currently theoretical scenario in which Bitcoin outputs (utxos) are accompanied by a proof of knowledge in the form of such a Schnorr signature.

2 Introduction

This note was motivated by the question of how one can and cannot embed data into the Bitcoin utxo set.

We emphasize that this focus removes the question of data embedded in the witness section of transactions, and OP_RETURN is also considered out of scope here, as that is utxo-set-prunable.
It is trivially possible to embed data into a single pubkey P, if the serialized key is only checked by checking its validity; approximately half of the 256 bit values between 1 and p (where p is the order of the secp256k1 underlying field) are x-coordinates that represent valid curve points. Thus one can easily embed close to 32 bytes of arbitrary data.

An obvious side point is that if the pubkey P is hashed, as it most commonly is for scriptPubKeys not using taproot, the same is true: the validity of the underlying pubkey here is not even checked, but as per the above it makes little difference, if the hash is also ~ 32 bytes.

The hypothetical scenario we here address is: what if we try to avoid that, by forcing the publisher to prove knowledge of the preimage of P? Would we thus prevent such data publishing being possible? The rest of this note attempts to answer that question.
As mentioned in the abstract, if some data is transferred via a sidechannel, this can certainly allow other data to be embedded into blockchain data. But from here, we focus in on the third case: attempts to embed data, without sidechannels, and without grinding:

Data can effectively be embedded in signatures by using a publically-inferrable nonce, as was noted here and was later fleshed out in detail here (**note**: both these sources discuss nonce-reuse but it's worse than that: any *publically inferrable* nonce can achieve the same thing, such as, the block hash of the parent block; this will have the same embedding rate and cannot be disallowed). This technique has a poor embedding rate (it's about 33% of the size of a tuple $(P, (R, s))$), because we leak one 32 byte value out of a total of 32+64 bytes) but is otherwise effective at combatting censorship. However it has another flaw that is crucial in our context: it allows the data to be immediately removed from the utxo set by adversaries (since the private key is published). In the remainder, we try to analyze and then formalize the significance of this.

2.1 Goal

Given the above context, we tighten our goal to: we wish to prove that embedding data into a combination (P, σ) where σ is a Schnorr pubkey-prefixed signature on the key P, is impossible beside grinding, if the owner wishes to keep control of the output (so as to keep it in the utxo set and/or spend it themselves at a later time).

We thus consider an adversary publishing outputs formatted as: (P, σ) with $\sigma = (R, s)$ a standard BIP340 Schnorr signature.

For simplicity we assume that the message signed over is "" (important note: since this signature type is pubkey prefixed, we already commit to the key (unlike in ECDSA signatures; see the final section of the document for more discussion on that).

And we try to answer the question: can you embed data at a more than trivial rate (through grinding, as is seen in e.g. vanity addresses) while creating such valid, signer-controlled outputs?

3 Proving unembeddability

The adversary wishing to leak data via a single output defined as (P, σ) will have to win the game PubUnEmbeddable$(\lambda, \mathbb{G}, H)$ (see Table 1). Some notes on its structure:

- the challenger will provide the message μ representing the data to be published. This models the idea that we are only really concerned about an adversary with the power to publish anything it chooses, rather than just some fixed message.
- the adversary must publish the function f before it receives the data μ; this is crucial mechanically of course: if it saw μ first and then could choose f, it would simply choose $f(..,..) = \mu$ and win. This represents the point that as per the Introduction, we want the data published to be readable without interaction with the readers; they simply need to know the "protocol rules" which are captured by f.
- The Adversary's response is likely surprising to the reader. He reveals the normally-private witness values (k, x) such that $R = kG$ and $P = xG$, and not a tuple (P, σ) as would be expected in a protocol to publish a key with an associated proof of knowledge (the Schnorr signature $\sigma = (R, s)$). Why is this? We assume the EUF-CMA security of Schnorr according to the normal proofs in the ROM of reducibility to ECDLP security. Thus $\nexists\ \mathcal{A}$ that can win a version of the game where $(P, (R, s))$ is returned, but cannot win this version of the game where (k, x) is returned.
- We represent the adversary's internal algorithm as A, a *deterministic* function on the provided μ, that outputs its choice of (k, x). We refer to this below as "one-pass" to emphasize that there is no grinding over multiple possibilities here.

Challenger	**Adversary**
	$\leftarrow f : (\mathbb{G}^2, \mathbb{Z}_n) \rightarrow \mathbb{Z}_n$
$\mu \leftarrow_\$ \mathcal{D}\ (\lvert\mathcal{D}\rvert = \omega(\lambda))$	
$\mu \rightarrow$	
	$\leftarrow (k, x) = A(\mu)$
Construct	
$R = kG,\ P = xG$	
$e = H(P, R, m ='')$	
$s = k + ex \mod n$	
and fail unless	
$f(P, R, s) = \mu$	

Table 1: Game PubUnEmbeddable(λ, $\mathbb{G}$, H)

Definition 1. A Schnorr signature scheme over an elliptic curve group $\mathbb{G}$ and hash fn H (with fixed null message) is **public-unembeddable** if, for every efficient adversary and every domain $\mathcal{D}$ of size superlogarithmic in the security parameter λ, the probability of winning the game PubUnEmbeddable$(\lambda, \mathbb{G}, H)$, without revealing the secret key x, is negligible.

Commentary:

- The reader is forgiven for confusion that the definition refers to "revealing the secret key x", when the game PubUnEmbeddable has the adversary literally hand over this key. Our intention is to demonstrate that the secret key is *publically calculable*, just from the transcript, P, R, s, however, in case the adversary wins the game.
- We require $|\mathcal{D}|$ to have superlogarithmic entropy in the security parameter λ. For smaller domains (e.g. polynomial in λ), an adversary can exhaustively search all possible μ.

Examples

The game PubUnEmbeddable can be won by choosing $k = g(x)$ to be any function of x such that $g(x) + ex$ is invertible. Here are some motivating examples:

- The trivial case: let $A(\mu) = (K(\mu), X(\mu))$. Let $X(\mu) = \mu$ and $K(\mu) = \mu$. This is the simplest possible way to encode μ via the secret key material. This X is the identity function, and is clearly invertible. We get the function $g(x) = x$ for k, also the identity function, and $s = k + ex = g(x) + ex = (1 + e)x$, which makes our $h_e(x)$ function trivially invertible: $x = \frac{s}{1+H(P,R,'')}$.
- The general affine case: If $g(x) = \alpha x + \beta$ with public $\alpha, \beta \in \mathbb{Z}_n$, then

$$s \equiv (\alpha + e)x + \beta \pmod{n}.$$

  For all $e \not\equiv -\alpha \pmod{n}$,

$$x = (s - \beta)(\alpha + e)^{-1} \pmod{n}, \qquad k = g(x).$$

  Thus both x and k are publicly extractable from one transcript. We note that due to the dependence of one variable on the other, this only represents the embedding of 256 bits of data, not 512.
- A quadratic function. Let $X(\mu) = \mu$ and $K(\mu) = a\mu^2 + b\mu + c$, that is, $g(x) = ax^2 + bx + c$. Then the function f will be:

$$f(P, R, s) = \left(-(b + e) \pm \sqrt{(b + e)^2 - 4a(c - s)}\right)/2a$$

  ... where $e = H(P, R,'')$. In this last case, notice that there are in general two solutions to the given quadratic, which cannot be fully controlled in advance since e is unavoidably random; however, it has been constructed so that $x = \mu$ is always *one* solution. Note also that is trivial to check if a candidate solution actually fits the signature; simply compare xG with P.

The above examples illustrate that **if** the nonce k is a function of x, $g(x)$, such that $g(x) + ex$ is invertible, **then** the game PubUnEmbeddable can be won. In the remainder we argue that there is no **other** way to win the game.

Theorem 1. *The BIP340 signature scheme is public-unembeddable in the ROM for SHA256.*

Lemma 1 (k and x are functionally related)**.** *If the adversary wins the game PubUnEmbeddable with non-negligible probability, then $k = g(x)$ is a function of x.*

Proof. Without explicitly referring to the forking lemma (and thus avoiding quantitative analysis), we can see that if we wrap the challenger and fork it at the calls to $e = H(P, R,'')$, then rewind and replay any successful attempt to win the game, we get:

$$f(P, R, s_1) = f(P, R, s_2) = \mu$$

where

$$s_1 = k + e_1 x, \quad s_2 = k + e_2 x$$

Thus if $k + e_1 x = k + e_2 x$, with e_1, e_2 chosen unpredictably at random by the oracle (and not limited to twice, of course), it follows that k must be a function of x (note that it would be equivalent to define x as a function of k [1]). □

Definition 2. A quick aside: we define **countably injective**, and will use this term, as the idea that a function can be reversed to find multiple possible solutions, but countably many (and computationally feasible to count). This definition is not standard but is important in practice. A simple example is that a general quadratic function f has an "inverse" that outputs in general two possible values, not one. This is distinct from a line like $3x + y$ which has infinite pairs x, y solving any specific output value.

Theorem 2 (Ability to win the game PubUnEmbeddable ⇒ Extraction)**.** *We assume Lemma 1. Then there exists a public function g such that $k = g(x)$ on the set of outputs of A ((k, x)). For any transcript (P, R, s) with $P = xG$, $R = g(x)G$, and $s = g(x) + ex \bmod n$, where $e = H(P\|R\|m_0)$, write*

$$s = h(x) := g(x) + ex$$

Then h^{-1} is derivable from f except with negligible probability, and

$$f = X^{-1} \circ h^{-1}$$

, and the secret key can be extracted as $h^{-1}(s)$.

Proof. The transcript satisfies $s = g(x) + ex$; we call $s = h(x)$.
h is countably injective on the domain Z_p: suppose it were not; then, we have $g(x_1) + e_1 x_1 = g(x_2) + e_2 x_2$, with $e-$values unpredictable outputs of the hash. **Assuming preimage resistance** (or, using the ROM) of the chosen cryptographic hash function H for $H(xG, kG,'') = e$, then we claim that h is effectively countably-injective, though note that this does not yet imply computability of h^{-1} on the codomain of h.

Since the adversary was required to produce $f(P, R, s) = \mu$, we can write $f(P, R, h(x)) = \mu$ and $f(P, R, h(X(\mu))) = \mu$ and so f is a left-inverse of the composite function $h \circ X$. Thus:

[1] This comment applies specifically to Schnorr, since the alternate forms $s = k + ex$ and $s = ek + x$ are functionally identical, differing only by a multiple, assuming that the inverse of H has the same properties as H itself. The comment does *not* apply to all signature schemes, and specifically does not apply to ECDSA.

$$f \circ h \circ X = \mathrm{id}$$
$$f \circ (h \circ X) \circ (h \circ X)^{-1} = (h \circ X)^{-1}$$
$$f = X^{-1} \circ h^{-1}$$

Thus we can find h^{-1} as $X \circ f$ and this must be computable if f is computable (we generally assume the trivial $X(\mu) = \mu$; other more complex relations could be folded into f, after all). □

Theorem 2 is thus proved assuming the preimage resistance of H, which directly implies Theorem 1 as the specific case of $\mathbb{G}$ = the group secp256k1 and H = SHA256.

4 Why this analysis does not apply to ECDSA

In the case of ECDSA, there is no intrinsic binding to the public key in the hash step. This means one can "back-solve" for the private key after having already passed the message through the hash function. This breaks the public-unembeddability property, as is proven in outline in the remainder:

Concretely, this corresponds to the reality that it is actually quite trivial in this "P-unbound" case to embed data into the signature, as was nicely illustrated in the second puzzle of Blockstream's half of a puzzle page. Just defer the choice of the private key to the end; fix s to your chosen data, fix $k, R = kG$, fix m, and solve for the private key: $d = \frac{(sk - H(m))}{R_x}$ (*note to reader: sk is a product of s and k!*). When you publish (R, s) you do not leak d.

Discursively: we consider again the idea of inverting functions. Let's start with Schnorr. Suppose your goal was to extract a secret key x from a given signature s; we have a function $f_k(x) = k + ex$ to go $x \rightarrow s$. If we want to go the other way, i.e. to invert the function, we have $x = \frac{s-k}{e}$. But because e depends on x you have a circular reference and cannot construct that function in reality if the hash function in e behaves cryptographically. While if you tried to apply this logic to ECDSA, it fails, because as already noted, the formula for the private key is: $d = \frac{(sk - H(m))}{R_x}$, which does not have any such circular dependency; note that there is an "implicit hash" of the nonce in R_x, but there is no such hash of the signing key.

If we try to apply the same formal model of a game PubUnEmbeddable2 (where the 2 refers to the ECDSA, instead of Schnorr, calculation performed by the challenger in the verification step), we find that the first step, that of Lemma 1 already fails. There is nowhere to fork; our hash function is now only taking constant input ". More specifically, we cannot fork on the choice of x by the adversary, at all. For this reason we cannot force the adversary to make k dependent on or related to x, and indeed, we have just seen that we can choose k entirely at random and thus hide the secret material. Casting things into the formal language, we can write $X(\mu) = \frac{\mu k - e}{r}$ with $e = H(m)$ and r the x-coord of kG. If we then want to collide on $X(\mu_1) = X(\mu_2)$, it is easy to arrange: just choose:

$$\mu_1 = \frac{e + \frac{r_1}{r_2}(\mu_2 k_2 - e)}{k_1}$$

This ability to arbitrarily collide without needing a violation of preimage resistance is just another way to illustrate that we cannot extract x from μ via a function inversion, and indeed that is what is seen in the previous half-a-puzzle example: the data is embedded without leaking the secret key.

We won't make a formal attempt to prove that one can recover the same security property with ECDSA, by requiring that the message m include P or a hash of P, though it seems likely that it exists. This seems uninteresting, since if a new Bitcoin consensus rule required P to be accompanied by σ, then it could always be required to be BIP340 rather than a legacy signature scheme with inferior security reductions.