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.

Theorem 1. *The BIP340 signature scheme is public-unembeddable if SHA256 is preimage resistant.*

Lemma 1 (Injectivity of X)**.** *Let A be single-pass: on input μ it deterministically outputs $(x, k) = (X(\mu), K(\mu))$. As per the security game, we posit the existence of an f such that $f(P, R, s) = \mu$ for all μ in a domain $\mathcal{D}$ of superlogarithmic size, where $P = xG$, $R = kG$, and $s = k + ex$ with $e = H(P, R, m_0)$. If such an f exists, i.e. the game PubUnEmbeddable can be won with non-negligible probability, then X must be injective on $\mathcal{D}$.*

(Note: we do not bother to mention discrete log hardness in $\mathbb{G}$ here; if discrete log is not hard, then Theorem 1 is vacuous: every signature would leak the secret witness, meaning the property public-unembeddable can never be achieved).

Proof. Let $A : \mathcal{D} \to \mathbb{Z}_n^2$, $A(\mu) = (X(\mu), K(\mu))$. Suppose that the adversary using A can win the game PubUnEmbeddable with non-negligible probability and that, for contradiction, A is not injective. Then there exist $\mu_1 \neq \mu_2$ such that

$$A(\mu_1) = A(\mu_2) = (x, k).$$

But the transcript (P, R, s) is fully determined by (x, k) via $P = xG$, $R = kG$, and $s = k + ex$. Hence both μ_1 and μ_2 yield identical transcripts. In that case no function $f(P, R, s)$ can distinguish whether the input was μ_1 or μ_2, contradicting the assumption that f correctly recovers every $\mu \in \mathcal{D}$ (which is the winning condition for PubUnEmbeddable). Therefore the function A used by a winning adversary must be injective. □

(Note that this argument is not dependent on Schnorr's algebraic structure $k+ex$ (whereas Theorem 2 below definitely *does* depend on that structure!), but only on the deterministic nature of the "public transcript" P, R, s; it is defined deterministically by the secret witness (k, x). Note also that the argument being "first-pass-only" is a way to model the idea that we cannot grind some bytes into the data by trial and error; if we allow this, then data can always be embedded, but since the work is exponential we feel at liberty to ignore this.)

Theorem 2 (Ability to win the game PubUnEmbeddable $\Rightarrow$ Extraction)**.** *Let the setting be as in 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 $e = H(P \| R \| m_0)$, define*

$$h_e(x) := g(x) + ex \pmod{n}.$$

For all but a negligible fraction of e, the map h_e is injective on the image of X. Hence there exists a public extractor

$$\mathrm{Ext}(P, R, s) = h_e^{-1}(s),$$

which recovers x (and then $k = g(x)$) from the single transcript.

Proof. By Lemma 1, X is injective on $\mathcal{D}$, so X has a well-defined inverse on its image $\mathcal{X} = X(\mathcal{D})$. Define $g(x) := K(X^{-1}(x))$. Then for each μ, A outputs $(x, g(x))$. The transcript satisfies $s = g(x) + ex$. Suppose two distinct $x_1 \neq x_2$ collide under h_e, i.e. $g(x_1) + ex_1 = g(x_2) + ex_2 \pmod{n}$. Then

$$e = (g(x_1) - g(x_2))(x_2 - x_1)^{-1} \pmod{n},$$

The probability that the hash function H output e hits such a bad value is negligible assuming the property of preimage resistance (otherwise an adversary might be able to find an e satisfying the above for a specific x_1, x_2 pair). Therefore, with overwhelming probability, h_e is injective on $\mathcal{X}$, so $x = h_e^{-1}(s)$ is uniquely determined. $\square$

(Actually, strict injectivity of h_e is not required. It suffices that, for each (e, s), the set $\{ x \in \mathcal{X} : h_e(x) = s \}$ is of size poly in λ and can thus be efficiently enumerated. In that case, the correct value of x is the unique element of this set satisfying $P = xG$, so extraction remains possible even if h_e is many-to-one not one-to-one. However the stricter statement seems to work fine, here.)

Remark. It is important to distinguish between *existence* of an inverse function and its *efficient computability.* In general, many injective functions $h : \mathcal{X} \to \mathbb{Z}_n$ possess a set-theoretic inverse h^{-1}, but there is no guarantee that this inverse can be evaluated in polynomial time. In fact, much of modern cryptography only exists because of this distinction!
But here we are working inside the game PubUnEmbeddable, where the adversary's chosen f is assumed to run in polynomial time. If such an f exists, then the relation $s = g(x) + ex$ implies that its inverse (what we call h, which returns x from s), must be efficiently computable.

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.

Examples

- The trivial case: 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.

3.1 Why this analysis does not apply directly 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, a claim we expand on, but don't fully prove, 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 follow the same formal modelling as used above for the Schnorr case, we can define the Unembeddability game the same way, and we can again conclude as in Lemma 1 that the mapping from μ to (x, k) must be injective, but we cannot make Theorem 2 go through: it requires us to form an inverse of the function $s = g(x)^{-1} (H(m) + R_x x)$, where here R_x is the x-coordinate of $R = g(x)G$ (this follows up the earlier commentary: Lemma 1 is not Schnorr-specific, but Theorem 2 is, as it requires us to form a function which only compounds the $k = g(x)$ relationship with a public scalar component, here e).

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.