BIP-340 (Schnorr) and BIP-341 (Taproot) are proposed upgrades to the Bitcoin network that create a new type of public key output which can be spent by (i) a Schnorr signature under that public key or (ii) revealing a hidden commitment to a script inside the public key and satisfying the conditions of the script. Framed as a hybrid commitment scheme:

\[ \text{MRC} \cdot \text{RPP} \rightarrow \text{TF} \]
\[ (\text{st. \ } m_1) \leftarrow \text{RPP} \]
\[ (G, X, T) \leftarrow \text{G} \]
\[ x \leftarrow Z_q \]
\[ y \leftarrow H(f(X)||m_1) \]
\[ i \leftarrow x \]
\[ L \leftarrow \{(G, 1, 0), (X, 0, 1), (T, y, 1)\} \]
\[ h_i \leftarrow \{1, 2, \ldots, q\} \]
\[ \left(x^i, m_2^i\right) \leftarrow \text{Agpp}(G, T, (X, m_1)) \]
\[ \text{return } m^i \]
\[ \text{return } L \cup \{(G, a_1, b_1)\} \]

If the hash function \( H \) is idealised as a random oracle then the scheme is secure[1]. Taking inspiration from [2], we instead idealise the elliptic curve group in the Generic Group Model to isolate what properties the hash function requires for Taproot to be secure. To compute new group elements the adversary is allowed up to \( q \) queries to the oracle \( G \) with two elements it already knows \((G_1, G_2)\). The oracle returns a new group element \( G_3 \) representing \( G_1 \cdot G_2 \). The main hash function properties we consider are:


\[ \text{RPP} \]
\[ (\text{st. } h) \leftarrow A \]
\[ P \leftarrow P \]
\[ m^i \leftarrow A(h, P) \]
\[ \text{return } H(P||m^i) = h \]

\[ \text{COPC} \]
\[ (\text{st. } h) \leftarrow A(P) \]
\[ P \leftarrow P \]
\[ (m_1, m_2) \leftarrow A(h, P) \]
\[ \text{return } H(P||m_1) - H(P||m_2) = \delta \]

Framed as a hybrid commitment scheme:

\[ \text{MRC} \cdot \text{RPP} \rightarrow \text{TF} \]
\[ (\text{st. } m_1) \leftarrow \text{RPP} \]
\[ (G, X, T) \leftarrow \text{G} \]
\[ x \leftarrow Z_q \]
\[ y \leftarrow H(f(X)||m_1) \]
\[ i \leftarrow x + y \]
\[ \text{return } x \]

\[ \text{return } (G, a_1, b_1) \]

If \( C \) is queried before \( X_2 \), then \( A \) clearly breaks collision resistance.

**Can an adversary come up with a covert Taproot spend by choosing their MuSig public key maliciously?** Call this the **MuSig Covert Taproot (MCT)** problem.

\[ \text{R}(P) : \text{COPC} \rightarrow \text{MSTC} \]
\[ x_i \leftarrow Z_q \]
\[ (G, X, T) \leftarrow \text{G} \]
\[ (h_i, t_i) \leftarrow \{(1, 2, \ldots, q)\} \]
\[ \text{return } L = \{(G, 1, 0), (X, 0, 1)\} \]
\[ (X_1, m_1) \leftarrow \text{Agpp}(G, T, (X, m_1)) \]
\[ \text{return } \left(x, X_1, C \cdot H(f(C)||m_1)\right) \]

**MuSig with Covert Taproot**

Can an adversary forge a fake opening on someone else’s coins? Call this the **Taproot Forge problem** (TF). RPP is necessary for TF to be hard:

\[ \text{TF} \]
\[ (\text{st. } m_1) \leftarrow \text{A} \]
\[ G \leftarrow \text{G} \]
\[ \left(x, X_1, m_1\right) \leftarrow A(\text{st. } G, \text{com}_A, \text{open}) \]
\[ \left(x, X_1, m_2\right) \leftarrow A(\text{st. } G, \text{com}_A, \text{open}) \]
\[ \text{return } X + H(f(X)||m_2) \]
\[ m_1, m_2 \neq m_1 \]
\[ \text{return } (C, m_2) \]

**MuSig Second Covert Taproot**

Can an adversary create a second malicious Taproot spend in addition to an agreed upon on one by choosing their parameters maliciously? Call this the **MuSig Second Covert Taproot (MSCT)** problem. COPC is necessary for MSCT to be hard:

\[ \text{MSCT} \]
\[ x_i \leftarrow G \]
\[ (X_1, m_1, (C, m_1)) \leftarrow A(X_i) \]
\[ X \leftarrow \text{MuSig}(X_1, X_2) \]
\[ \text{return } X \]

**Forging an Opening**

RPP is sufficient to ensure MCT is hard if \( X_2 \) is queried before \( C \). If the reduction guesses correctly which queries will be used for \( X_2 \) and \( C \) solves RPP. This approach only works for 2-party MuSig.

\[ \text{R} : \text{RPP} \rightarrow \text{MCT} \]
\[ x_i \leftarrow Z_q \]
\[ (G, X, T) \leftarrow \text{G} \]
\[ (h_i, t_i) \leftarrow \{(1, 2, \ldots, q)\} \]
\[ \text{return } L = \{(G, 1, 0), (X, 0, 1)\} \]
\[ (X_1, m_1) \leftarrow \text{Agpp}(G, T, (X, m_1)) \]
\[ \text{return } \left(x, X_1, C \cdot H(f(C)||m_1)\right) \]

**Remarks**

- These reductions are incomplete - they do not account for \( A \) choosing \( G \) or \( X_1 \) etc as one of the elements they return. They can be modified to fix this.
- To actually steal coins, the malicious Taproot openings have to be valid Merkle Root (\( m \) can’t be arbitrary).
- If coin tossing is used to generate joint key instead of MuSig then security in all scenarios follows from RPP.