

A Note on Unforgeability of MuSig2 with Tweaking

Jonas Nick¹, Tim Ruffing¹, and Yannick Seurin²

¹ Blockstream

² ANSSI, Paris, France

Abstract. *Key Tweaking* refers to the process of producing a new pair of secret and public key from a given pair. This is used, for example, to derive fresh keys from a master keypair or to commit to a value in a public key. In this note, we show that a variant of MuSig2 with naive tweaking is insecure and propose a variant that is not vulnerable against the attack.

1 The Vulnerable Scheme

This is MuSig with some version of tweaking. weak because attacker only has control over the contract, not the tweak. TODO: what is tweaking See Figure 2.

2 Generalized Birthday Problem

The attack against MuSig2NaiveTweak makes use of Wagner’s algorithm for solving the Generalized Birthday Problem It can be defined as follows for the purpose of this paper: Given a constant value $t \in \mathbb{Z}_p$, an integer k_{\max} , and access to random oracle H mapping onto \mathbb{Z}_p , find a set $\{q_1, \dots, q_{k_{\max}}\}$ of k_{\max} queries such that $\sum_{k=1}^{k_{\max}} H(q_k) = t$. For $k_{\max} = 2^{\sqrt{\log_2(p)}-1}$ the complexity of this algorithm is $O(2^{2\sqrt{\log_2(p)}})$.

Jonas’ note: Perhaps can use BLOR? “If the attacker is able to open more sessions concurrently, the improved polynomial-time attack by Benhamouda *et al.* [add:BLOR20] assumes $k_{\max} > \log_2 p$ sessions, but then has complexity $O(k_{\max} \log_2 p)$ and a negligible running time in practice.”

3 Description of the Attack against MuSig2NaiveTweak

The adversary calls KeyTweak $\ell_{\max} \in O(2^{2\sqrt{\log_2(p)}})$ times to obtain values $t^{(1)}, \dots, t^{(\ell_{\max})}$ and computes the multiset of public keys L and aggregate key \tilde{X} for the (untweaked) public key of the honest signer $X'_1 = g^{x_1}$ as

$$L = \{X'_1 g^{t^{(1)}}, \dots, X'_1 g^{t^{(\ell_{\max})}}\}$$

$$\tilde{X} = \text{KeyAgg}(L).$$

Then, the adversary opens $k_{\max} = 2^{\sqrt{\log_2(p)}-1}$ concurrent signing sessions by requesting k_{\max} nonce tuples $R_{1,1}^{(1)}, \dots, R_{1,\nu}^{(1)}, \dots, R_{1,1}^{(k_{\max})}, \dots, R_{1,\nu}^{(k_{\max})}$ from the honest signer and computes

$$R_j = \sum_{k=1}^{k_{\max}} R_{1,j}^{(k)}, \quad j \in [1, \nu]$$

$$b = H_{\text{non}}(\tilde{X}, (R_1, \dots, R_\nu), m)$$

$$R^* = \prod_{j=1}^{\nu} R_j^{b^{j-1}}.$$

Now it is possible to use Wagner’s algorithm to find a function $f : [1, k_{\max}] \rightarrow [1, \ell_{\max}]$ that associates a value $t^{(\ell)}$ to each session k such that

Setup (1^λ) <hr/> $(\mathbb{G}, p, g) \leftarrow \text{GrGen}(1^\lambda)$ Select three hash functions $H_{\text{agg}}, H_{\text{non}}, H_{\text{sig}} : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ $\text{par} := ((\mathbb{G}, p, g), H_{\text{agg}}, H_{\text{non}}, H_{\text{sig}})$ return par	SignAgg ($\text{out}_1, \dots, \text{out}_n$) <hr/> for $i := 1 \dots n$ do $(R_{i,1}, \dots, R_{i,\nu}) := \text{out}_i$ for $j := 1 \dots \nu$ do $R_j := \prod_{i=1}^n R_{i,j}$ return $\text{out} := (R_1, \dots, R_\nu)$
KeyGen () <hr/> $x \leftarrow \$\mathbb{Z}_p$; $X := g^x$ $sk := x$; $pk := X$ return (sk, pk)	Sign' ($\text{state}_1, \text{out}, sk_1, m, (pk_2, \dots, pk_n), \mathbf{t}$) <hr/> $\text{// Sign' must be called at most once per } \text{state}_1.$ if \mathbf{t} has not been output by $\mathbf{T}_{\text{tweaks}}$ then return false $(r_{1,1}, \dots, r_{1,\nu}) := \text{state}_1$ $x_1 := sk_1 + \mathbf{t}$; $X_1 := g^{x_1 + \mathbf{t}}$ $(X_2, \dots, X_n) := (pk_2, \dots, pk_n)$ $L := \{X_1, \dots, X_n\}$ $a_1 := \text{KeyAggCoef}(L, X_1)$ $\tilde{X} := \text{KeyAgg}(L)$ $(R_1, \dots, R_\nu) := \text{out}$ $b := H_{\text{non}}(\tilde{X}, (R_1, \dots, R_\nu), m)$ $R := \prod_{j=1}^\nu R_j^{b^{j-1}}$ $c := H_{\text{sig}}(\tilde{X}, R, m)$ $s_1 := ca_1 x_1 + \sum_{i=1}^\nu r_{1,i} b^{i-1} \bmod p$ $\text{state}'_1 := R$; $\text{out}'_1 := s_1$ return $(\text{state}'_1, \text{out}'_1)$
KeyTweak () <hr/> $\mathbf{t} \leftarrow \$\mathbb{Z}_p$ return \mathbf{t}	
KeyAggCoef (L, X_i) <hr/> return $H_{\text{agg}}(L, X_i)$	
KeyAgg (L) <hr/> $\{X_1, \dots, X_n\} := L$ for $i := 1 \dots n$ do $a_i := \text{KeyAggCoef}(L, X_i)$ return $\tilde{X} := \prod_{i=1}^n X_i^{a_i}$	
Ver (\tilde{pk}, m, σ) <hr/> $\tilde{X} := \tilde{pk}$; $(R, s) := \sigma$ $c := H_{\text{sig}}(\tilde{X}, R, m)$ return $(g^s = R\tilde{X}^c)$	SignAgg' ($\text{out}'_1, \dots, \text{out}'_n$) <hr/> $(s_1, \dots, s_n) := (\text{out}'_1, \dots, \text{out}'_n)$ $s := \sum_{i=1}^n s_i \bmod p$ return $\text{out}' := s$
Sign () <hr/> $\text{// Local signer has index 1.}$ for $j := 1 \dots \nu$ do $r_{1,j} \leftarrow \$\mathbb{Z}_p$; $R_{1,j} := g^{r_{1,j}}$ $\text{out}_1 := (R_{1,1}, \dots, R_{1,\nu})$ $\text{state}_1 := (r_{1,1}, \dots, r_{1,\nu})$ return $(\text{out}_1, \text{state}_1)$	Sign'' ($\text{state}'_1, \text{out}'$) <hr/> $R := \text{state}'_1$; $s := \text{out}'$ return $\sigma := (R, s)$

Fig. 1. The multi-signature scheme $\text{MuSig2NaiveTweak}[\text{GrGen}, \nu]$. The differences to $\text{MuSig2}[\text{GrGen}, \nu]$ are displayed in **red**.

$$\sum_{k=1}^{k_{\max}} \underbrace{\text{H}_{\text{agg}}(L, X'_1 g^{t^{(f(k))}})}_{=: a_1^{(k)}} \underbrace{\text{H}_{\text{sig}}(\tilde{X}, R^*, m)}_{=: c^{(k)}} = \underbrace{\text{H}_{\text{sig}}(X'_1, R^*, m^*)}_{=: c^*}. \quad (1)$$

for a forgery target message m^* . For all $k \in [1, k_{\max}]$ the honest signer is asked for a partial signature using value $C^{f(k)}$ which is answered with $s_1^{(k)} = r_{1,1}^{(k)} + br_{1,2}^{(k)} + c^{(k)} \cdot a_1^{(k)}(x_1 + t^{(f(k))})$. This allows the adversary to compute

$$s_1^{*'} = \sum_{k=1}^{k_{\max}} s_1^{(k)} \quad (2)$$

$$= \sum_{k=1}^{k_{\max}} r_{1,1}^{(k)} br_{1,2}^{(k)} + \left(\sum_{k=1}^{k_{\max}} c^{(k)} a_1^{(k)} \right) \cdot x_1 + \sum_{k=1}^{k_{\max}} c^{(k)} a_1^{(k)} t^{(f(k))} \quad (3)$$

$$= \log_g(R^*) + c^* x_1 + \sum_{k=1}^{k_{\max}} c^{(k)} a_1^{(k)} t^{(f(k))} \quad (4)$$

where the last equality follows from Equation (1). The last summand can be subtracted as

$$s_1^* = s_1^{*'} - \sum_{k=1}^{k_{\max}} c^{(k)} a_1^{(k)} t^{(f(k))}$$

to obtain (R^*, s^*) , a valid forgery on message m^* for public key X'_1 .

4 Where the security proof of MuSig2 fails against MuSig2NaiveTweak

TODO Look at ROM proof of MuSig (section) We have a != a but b = b

5 Tweaking in MuSig2 without

TODO Section's bla and blub indicate that is secure when Make sure that attacker can not choose the signers pubkey after seeing the signers nonces. Here, adversary can have full control over t .

Other possible fixes:

- commit to pk
- use separate b per signer

6 Conclusion

Other multisigs and fix?

- MuSig1 with naive tweaking
- multisig with PoK and naive tweaking: there's no “naive” tweaking with PoK
- FROST-1 with naive tweaking: not vulnerable due to different b_i
- FROST-2: perhaps vulnerable due to lagrange coefficients, need to input set of signers into nonce gen (but probable need to do that anyway)

<div style="border-bottom: 1px solid black; padding-bottom: 5px;"> Setup(1^λ) </div> $(\mathbb{G}, p, g) \leftarrow \text{GrGen}(1^\lambda)$ <p>Select three hash functions</p> $H_{\text{agg}}, H_{\text{non}}, H_{\text{sig}} : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ $par := ((\mathbb{G}, p, g), H_{\text{agg}}, H_{\text{non}}, H_{\text{sig}})$ <p>return par</p> <div style="border-bottom: 1px solid black; padding-bottom: 5px;"> KeyGen() </div> $x \leftarrow \$\mathbb{Z}_p; X := g^x$ $sk := x; pk := X$ <p>return (sk, pk)</p> <div style="border-bottom: 1px solid black; padding-bottom: 5px;"> KeyTweak() </div> $t \leftarrow \$\mathbb{Z}_p$ <p>return t</p> <div style="border-bottom: 1px solid black; padding-bottom: 5px;"> KeyAggCoef(L, X_i) </div> <p>return $H_{\text{agg}}(L, X_i)$</p> <div style="border-bottom: 1px solid black; padding-bottom: 5px;"> KeyAgg(L) </div> $\{X_1, \dots, X_n\} := L$ <p>for $i := 1 \dots n$ do</p> $a_i := \text{KeyAggCoef}(L, X_i)$ <p>return $\tilde{X} := \prod_{i=1}^n X_i^{a_i}$</p> <div style="border-bottom: 1px solid black; padding-bottom: 5px;"> Ver(\tilde{pk}, m, σ) </div> $\tilde{X} := \tilde{pk}; (R, s) := \sigma$ $c := H_{\text{sig}}(\tilde{X}, R, m)$ <p>return $(g^s = R\tilde{X}^c)$</p> <div style="border-bottom: 1px solid black; padding-bottom: 5px;"> Sign(sk_1) </div> $t := \text{KeyTweak}()$ <p>// Local signer has index 1.</p> <p>for $j := 1 \dots \nu$ do</p> $r_{1,j} \leftarrow \$\mathbb{Z}_p; R_{1,j} := g^{r_{1,j}}$ $out_1 := (R_{1,1}, \dots, R_{1,\nu})$ $state_1 := (r_{1,1}, \dots, r_{1,\nu}, sk_1 + t \bmod p)$ <p>return $(out_1, state_1)$</p>	<div style="border-bottom: 1px solid black; padding-bottom: 5px;"> SignAgg(out_1, \dots, out_n) </div> <p>for $i := 1 \dots n$ do</p> $(R_{i,1}, \dots, R_{i,\nu}) := out_i$ <p>for $j := 1 \dots \nu$ do</p> $R_j := \prod_{i=1}^n R_{i,j}$ <p>return $out := (R_1, \dots, R_\nu)$</p> <div style="border-bottom: 1px solid black; padding-bottom: 5px;"> Sign'($state_1, out, \cancel{sk_1}, m, (pk_2, \dots, pk_n)$) </div> <p>// Sign' must be called at most once per $state_1$.</p> $(r_{1,1}, \dots, r_{1,\nu}, \mathbf{x}_1) := state_1$ $X_1 := g^{x_1}$ $(X_2, \dots, X_n) := (pk_2, \dots, pk_n)$ $L := \{X_1, \dots, X_n\}$ $a_1 := \text{KeyAggCoef}(L, X_1)$ $\tilde{X} := \text{KeyAgg}(L)$ $(R_1, \dots, R_\nu) := out$ $b := H_{\text{non}}(\tilde{X}, (R_1, \dots, R_\nu), m)$ $R := \prod_{j=1}^\nu R_j^{b^{j-1}}$ $c := H_{\text{sig}}(\tilde{X}, R, m)$ $s_1 := ca_1x_1 + \sum_{i=1}^\nu r_{1,i}b^{i-1} \bmod p$ $state'_1 := R; out'_1 := s_1$ <p>return $(state'_1, out'_1)$</p> <div style="border-bottom: 1px solid black; padding-bottom: 5px;"> SignAgg'(out'_1, \dots, out'_n) </div> $(s_1, \dots, s_n) := (out'_1, \dots, out'_n)$ $s := \sum_{i=1}^n s_i \bmod p$ <p>return $out' := s$</p> <div style="border-bottom: 1px solid black; padding-bottom: 5px;"> Sign''($state'_1, out'$) </div> $R := state'_1; s := out'$ <p>return $\sigma := (R, s)$</p>
---	---

Fig. 2. The multi-signature scheme $\text{MuSig2Tweak}[\text{GrGen}, \nu]$. The differences to $\text{MuSig2}[\text{GrGen}, \nu]$ are displayed in **red**.