

# A Note on Unforgeability of MuSig2 with Key Tweaking

No Author Given

No Institute Given

**Abstract.** *Key Tweaking* refers to the process of producing a new pair of secret and public key from a given keypair. This is used, for example, to derive fresh keys from a master keypair or to create a commitment to a value such that the commitment is also a public key. In this note, we demonstrate that a variant of MuSig2 with naive support for key tweaking is insecure and show an alternative method of tweaking that is not vulnerable to the attack.

## 1 The Vulnerable Scheme

Figure 1 shows MuSig2NaiveTweak, a multi-signature scheme identical to MuSig2 [NRS21] except that it additionally allows signing for a tweaked public key. There are multiple variants of tweaking, which differ mainly in how the tweak  $t$  is derived. We chose a variant of tweaking for MuSig2NaiveTweak that gives the adversary the minimal capability necessary to execute the attack. In particular, the honest MuSig2NaiveTweak signer has an algorithm **KeyTweak** that generates public, uniformly random tweaks without input from the adversary. The signer accepts an externally provided tweak when signing but only if it has been output by **KeyTweak**.

MuSig2NaiveTweak uses *additive* tweaking, but the attack similarly applies to *multiplicative* tweaking.

## 2 Generalized Birthday Problem

The attack against MuSig2NaiveTweak described below makes use of Wagner’s algorithm [Wag02] for solving the Generalized Birthday Problem. It can be defined as follows for the purpose of this note: 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  $O(2^{2\sqrt{\log_2(p)}})$ .

## 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

$$R_j = \sum_{k=1}^{k_{\max}} R_{1,j}^{(k)}, \quad j \in [1, \nu]$$
$$b = H_{\text{non}}(\tilde{X}, (R_1, \dots, R_\nu), m)$$
$$R^* = \prod_{j=1}^{\nu} R_j^{b^{j-1}}.$$

<b>Setup</b> ( $1^\lambda$ ) <hr/> $(\mathbb{G}, p, g) \leftarrow \text{GrGen}(1^\lambda)$ Select three hash functions $H_{\text{agg}}, H_{\text{non}}, H_{\text{sig}} : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ $par := ((\mathbb{G}, p, g), H_{\text{agg}}, H_{\text{non}}, H_{\text{sig}})$ <b>return</b> $par$	<b>SignAgg</b> ( $out_1, \dots, out_n$ ) <hr/> <b>for</b> $i := 1 \dots n$ <b>do</b> $(R_{i,1}, \dots, R_{i,\nu}) := out_i$ <b>for</b> $j := 1 \dots \nu$ <b>do</b> $R_j := \prod_{i=1}^n R_{i,j}$ <b>return</b> $out := (R_1, \dots, R_\nu)$
<b>KeyGen</b> () <hr/> $x \leftarrow \$ \mathbb{Z}_p$ ; $X := g^x$ $sk := x$ ; $pk := X$ <b>return</b> $(sk, pk)$	<b>Sign'</b> ( $state_1, out, sk_1, m, (pk_2, \dots, pk_n), \mathbf{t}$ ) <hr/> // Sign' must be called at most once per $state_1$ . <b>if</b> $\mathbf{t} \neq \mathbf{0}$ and has not been output by <b>KeyTweak</b> <b>then return false</b> $(r_{1,1}, \dots, r_{1,\nu}) := state_1$ $x_1 := sk_1 + \mathbf{t} \bmod p$ ; $X_1 := g^{x_1}$ $(X_2, \dots, X_n) := (pk_2, \dots, pk_n)$ $L := \{X_1, \dots, X_n\}$ $a_1 := \text{KeyAggCoef}(L, X_1)$ $\tilde{X} := \text{KeyAgg}(L)$ $(R_1, \dots, R_\nu) := out$ $b := H_{\text{non}}(\tilde{X}, (R_1, \dots, R_\nu), m)$ $R := \prod_{j=1}^\nu R_j^{b^{j-1}}$ $c := H_{\text{sig}}(\tilde{X}, R, m)$ $s_1 := ca_1 x_1 + \sum_{i=1}^\nu r_{1,i} b^{i-1} \bmod p$ $state'_1 := R$ ; $out'_1 := s_1$ <b>return</b> $(state'_1, out'_1)$
<b>KeyTweak</b> () <hr/> $\mathbf{t} \leftarrow \$ \mathbb{Z}_p$ <b>return</b> $\mathbf{t}$	
<b>KeyAggCoef</b> ( $L, X_i$ ) <hr/> <b>return</b> $H_{\text{agg}}(L, X_i)$	
<b>KeyAgg</b> ( $L$ ) <hr/> $\{X_1, \dots, X_n\} := L$ <b>for</b> $i := 1 \dots n$ <b>do</b> $a_i := \text{KeyAggCoef}(L, X_i)$ <b>return</b> $\tilde{X} := \prod_{i=1}^n X_i^{a_i}$	
<b>Ver</b> ( $\tilde{pk}, m, \sigma$ ) <hr/> $\tilde{X} := \tilde{pk}$ ; $(R, s) := \sigma$ $c := H_{\text{sig}}(\tilde{X}, R, m)$ <b>return</b> $(g^s = R\tilde{X}^c)$	<b>SignAgg'</b> ( $out'_1, \dots, out'_n$ ) <hr/> $(s_1, \dots, s_n) := (out'_1, \dots, out'_n)$ $s := \sum_{i=1}^n s_i \bmod p$ <b>return</b> $out' := s$
<b>Sign</b> () <hr/> // Local signer has index 1. <b>for</b> $j := 1 \dots \nu$ <b>do</b> $r_{1,j} \leftarrow \$ \mathbb{Z}_p$ ; $R_{1,j} := g^{r_{1,j}}$ $out_1 := (R_{1,1}, \dots, R_{1,\nu})$ $state_1 := (r_{1,1}, \dots, r_{1,\nu})$ <b>return</b> $(out_1, state_1)$	<b>Sign''</b> ( $state'_1, out'$ ) <hr/> $R := state'_1$ ; $s := out'$ <b>return</b> $\sigma := (R, s)$

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

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

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

for a forgery target message  $m^*$ . For all  $k \in [1, k_{\max}]$ , the adversary asks the honest signer for a partial signature using value  $t^{(f(k))}$  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 adversary can subtract the last summand

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

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

## 4 BLLOR Attack

Benhamouda, Lepoint, Loss, Orrù, and Raykova [BLL+21] describe an algorithm that solves the ROS problem and can be applied to attack to break the unforgeability of **MuSig2NaiveTweak**. If the adversary can open at least  $\log_2 p$  sessions, then the algorithm has complexity  $O(\log_2^2 p)$  and a negligible running time in practice (otherwise, a variant of the algorithm can be applied that has a higher complexity). In contrast to the attack based on Wagner's algorithm, this attack allows using multisets of public keys with only two elements.

## 5 Where the Security Proof of **MuSig2** Fails when Adding Naive Tweaking

**Jonas' note:** TODO: In the ROM proof we would have two exections where  $a \neq a'$  but  $b = b'$ .

## 6 A Fixed **MuSig2** Variant with Tweaking

The attack in [section 3](#) relies on selecting a tweak  $t^{(k)}$  for a signing session  $k$  *after* obtaining nonces  $R_{1,1}^{(k)}, \dots, R_{1,\nu}^{(k)}$ . If  $t^{(k)}$  was determined before the nonces of the session, then the function  $f$  in Equation (1) is fixed and can not be influenced by the attacker. This is accomplished by the scheme shown in [Figure 1](#), which associates a fixed tweak when generating the nonces and is, therefore, not vulnerable to the attack.

Instead of generating the tweak in **Sign** as shown in [Figure 1](#), one can also prevent the attack by modifying **Sign** to take the tweaked secret key  $sk_1 + t^{(k)} \bmod p$  or tweaked public key  $X_1 g^{t^{(k)}}$  as input. This modified scheme would then write the tweaked secret or public key into  $state_1$  and make sure that **Sign'**( $state_1, \dots$ ) outputs a signature for the tweaked secret or public key contained in  $state_1$ .

It is also possible to stop the attack without having to determine tweak  $t^{(k)}$  before outputting the session's nonces. This can be achieved by changing the scheme so that each signer  $i$  gets a different nonce coefficient  $b_i$  instead of using a single nonce coefficient  $b$  for all signers of a signing session. However, using a separate nonce coefficient  $b_i$  increases the communication complexity of the scheme because it prevents nonce aggregation via **SignAgg**. In this scheme, all other signers' nonces  $R_{2,1}, \dots, R_{2,\nu}, \dots, R_{n,1}, \dots, R_{n,\nu}$  are required to be input to **Sign** and  $b_i$ ,  $R$  and  $s_1$  are computed as

$$\begin{aligned} b_i &:= H_{\text{non}}(\tilde{X}, (\mathbf{R}_{1,1}, \dots, \mathbf{R}_{1,\nu}, \dots, \mathbf{R}_{n,1}, \dots, \mathbf{R}_{n,\nu}), m, \mathbf{X}_i) \quad i \in [1, n] \\ R &:= \prod_{i=1}^n \prod_{j=1}^{\nu} R_{i,j}^{b_i^{j-1}} \\ s_1 &:= ca_1 x_1 + \sum_{i=1}^{\nu} r_{1,i} b_1^{i-1} \bmod p. \end{aligned}$$

## 7 Which Schemes are Vulnerable?

The attack discussed in this note targets the key aggregation coefficient  $a_1$  in **MuSig2**. **MuSig1** would be similarly vulnerable if the adversary can select the tweak to sign with after seeing the nonce. Schemes without a key aggregation coefficient, e.g., those relying on proof-of-knowledge of the public key instead of **MuSig**-style key aggregation, are not affected.

## Acknowledgments

We thank Yannick Seurin for identifying a vulnerability in a related scheme that ultimately led to the discovery of the attack discussed in this note.

## References

- [BLL+21] F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, and M. Raykova. “On the (in)security of ROS”. In: *EUROCRYPT 2021*. 2021.
- [NRS21] J. Nick, T. Ruffing, and Y. Seurin. “MuSig2: simple two-round Schnorr multi-signatures”. In: *Annual International Cryptology Conference*. Springer. 2021, pp. 189–221.
- [Wag02] D. Wagner. “A Generalized Birthday Problem”. In: 2002, pp. 288–303. DOI: [10.1007/3-540-45708-9\\_19](https://doi.org/10.1007/3-540-45708-9_19).

<b>Setup</b> ( $1^\lambda$ ) <hr/> $(\mathbb{G}, p, g) \leftarrow \text{GrGen}(1^\lambda)$ Select three hash functions $H_{\text{agg}}, H_{\text{non}}, H_{\text{sig}} : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ $par := ((\mathbb{G}, p, g), H_{\text{agg}}, H_{\text{non}}, H_{\text{sig}})$ <b>return</b> $par$	<b>SignAgg</b> ( $out_1, \dots, out_n$ ) <hr/> <b>for</b> $i := 1 \dots n$ <b>do</b> $(R_{i,1}, \dots, R_{i,\nu}) := out_i$ <b>for</b> $j := 1 \dots \nu$ <b>do</b> $R_j := \prod_{i=1}^n R_{i,j}$ <b>return</b> $out := (R_1, \dots, R_\nu)$
<b>KeyGen</b> () <hr/> $x \leftarrow \$ \mathbb{Z}_p$ ; $X := g^x$ $sk := x$ ; $pk := X$ <b>return</b> $(sk, pk)$	<b>Sign'</b> ( $state_1, out, sk_1, m, (pk_2, \dots, pk_n)$ ) <hr/> $\text{// Sign' must be called at most once per } state_1.$ $(r_{1,1}, \dots, r_{1,\nu}, \mathbf{t}) := state_1$ $x_1 := sk_1 + \mathbf{t} \bmod p$ ; $X_1 := g^{x_1}$ $(X_2, \dots, X_n) := (pk_2, \dots, pk_n)$ $L := \{X_1, \dots, X_n\}$ $a_1 := \text{KeyAggCoef}(L, X_1)$ $\tilde{X} := \text{KeyAgg}(L)$ $(R_1, \dots, R_\nu) := out$ $b := H_{\text{non}}(\tilde{X}, (R_1, \dots, R_\nu), m)$ $R := \prod_{j=1}^\nu R_j^{b^{j-1}}$ $c := H_{\text{sig}}(\tilde{X}, R, m)$ $s_1 := ca_1 x_1 + \sum_{i=1}^\nu r_{1,i} b^{i-1} \bmod p$ $state'_1 := R$ ; $out'_1 := s_1$ <b>return</b> $(state'_1, out'_1)$
<b>KeyTweak</b> () <hr/> $\mathbf{t} \leftarrow \$ \mathbb{Z}_p$ <b>return</b> $\mathbf{t}$	<b>SignAgg'</b> ( $out'_1, \dots, out'_n$ ) <hr/> $(s_1, \dots, s_n) := (out'_1, \dots, out'_n)$ $s := \sum_{i=1}^n s_i \bmod p$ <b>return</b> $out' := s$
<b>KeyAggCoef</b> ( $L, X_i$ ) <hr/> <b>return</b> $H_{\text{agg}}(L, X_i)$	<b>Sign''</b> ( $state'_1, out'$ ) <hr/> $R := state'_1$ ; $s := out'$ <b>return</b> $\sigma := (R, s)$
<b>KeyAgg</b> ( $L$ ) <hr/> $\{X_1, \dots, X_n\} := L$ <b>for</b> $i := 1 \dots n$ <b>do</b> $a_i := \text{KeyAggCoef}(L, X_i)$ <b>return</b> $\tilde{X} := \prod_{i=1}^n X_i^{a_i}$	
<b>Ver</b> ( $\tilde{pk}, m, \sigma$ ) <hr/> $\tilde{X} := \tilde{pk}$ ; $(R, s) := \sigma$ $c := H_{\text{sig}}(\tilde{X}, R, m)$ <b>return</b> $(g^s = R\tilde{X}^c)$	
<b>Sign</b> () <hr/> $\mathbf{t} := \text{KeyTweak}()$ $\text{// Local signer has index 1.}$ <b>for</b> $j := 1 \dots \nu$ <b>do</b> $r_{1,j} \leftarrow \$ \mathbb{Z}_p$ ; $R_{1,j} := g^{r_{1,j}}$ $out_1 := (R_{1,1}, \dots, R_{1,\nu})$ $state_1 := (r_{1,1}, \dots, r_{1,\nu}, \mathbf{t})$ <b>return</b> $(out_1, state_1)$	

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