A short note on signature adaptor security

Status: DRAFT

May 7, 2023

1 Overview

In this note, we give a definition of a “signature adaptor” (first defined by Poelstra [1]) in terms of the Schnorr signature (specifically the key-prefixed version standardized in BIP340 [4]), starting with a “toy” definition, for a single public key P.

We then state formally the security properties of this “toy” construction, before explaining why it is practically useless. Nevertheless, this forms the basis of the remainder.

Next we explain how such a “signature adaptor” can be incorporated into a 3-round MuSig [2] protocol (**note**: all references to MuSig from now on refer to the construction in that paper; in particular, not MuSig2 [3]). for signing over an aggregated key (P_{agg}), and explain how its security properties relate to the previous case.

Then, we address the scenario of the use of multiple signature adaptors *concurrently* in a MuSig signing session. In particular, we address the probably most important case: where the *same* adaptor point (and therefore secret) is being used concurrently in multiple different signing sessions. We find that security of such a construction can be proved by a reduction argument, but only with some important caveats. Further work in this area is likely needed.

This last analysis is useful because it applies to the most well known constructions that require signature adaptors, namely the adaptor based CoinSwap [14] [15], as well as more exotic versions like “Multiparty S6” [6].

2 Notation

Although we are tacitly assuming the use of the elliptic curve secp256k1 (see next paragraph), the remainder should apply to any other group of prime order in which the discrete log problem is hard.
We use $\mathbb{H}$ to mean specifically the hash function defined in BIP340 [4] for hashing messages (though of course, other choices are possible). We use $\mathbb{H}_c$ to mean the hash function for committing to curve points, in the unmodified and modified versions of MuSig discussed here.
Where we refer to the creation of nonces, we always assume “chosen at random”, i.e. there is no notion of “deterministic random”, the security or insecurity of

which is discussed at length in BIP340 and the MuSig2 BIP [5], amongst other places.
Additionally, we will always use R for signature nonce points, and their corresponding scalar values will be k (with appropriate sub/superscripts).

3 Earlier Work on Adaptor Security

Before going on to our base definitions, which set the stage for our set of security arguments, it is worth addressing how security of signature adaptors is treated in other works.

As far as we know, the first formal treatments of the security properties of signature adaptors were found in Fournier [13] and in Aumayr et al. [9]. Focusing first on the latter, the concept of aEUF-CMA security is defined in terms of an adversary with access to a normal signing oracle as well as an adaptor signing oracle. Definition 1 of that paper argues for unforgeability in terms of this adversary, who is given a single adaptor on the message to be forged. We would claim that this definition is a little too weak, since it is more interesting to consider the case where the adversary has access to *multiple* signature adaptors on the *one* message for which a full signature forgery is to be produced. As a critique of the model, this is debatable, however this point is at the heart of what is discussed in the rest of this paper.

This same security model was also adopted for work on payment channels in [12]. Many of these papers need a generic construction, not specific to Schnorr, not least because they consider the possibility of instantiating signature adaptors using the ECDSA instead of the Schnorr primitive - a setup which is substantially more cryptographically complex.

Also relevant, from [9], is the claim that signature adaptors (there called "pre-signatures") are unforgeable. However there is vital nuance assumed in that statement, which is proved formally in Appendix D: an adversary cannot forge an adaptor on a *given* "statement" T (in our case, a curve point or public key). This says nothing about the ability of the adversary to create such a forgery on a random, not pre-chosen T; and indeed, this ability will be central to the below arguments. (Also, it can hardly be considered out of scope in practical protocols, since very often these T values will be chosen randomly).

A closely related point is made by Fournier in [13]; to quote: *EUF-CMA[VES] says nothing about the unforgeability of signature encryptions. In fact, an adversary who can produce valid VES ciphertexts without the secret signing key is perfectly compatible. Of course, they will never be able to forge a VES ciphertext under a particular encryption key. If they could do that, then they could trivially forge an encrypted signature under a key for which they know the decryption key and decrypt it.*. Indeed, as the paper later alludes in Section 3, there may be protocols in which a signature adaptor is created by another party in a multi-party collaborative protocol. In Fournier's model, he treats signature adaptors as "one time Verifiably Encrypted Signatures" (VES), with "one time" of course referring to the property that the revelation of the plaintext, with ciphertext already known, reveals the secret key (i.e. the discrete log for the Schnorr case, the t corresponding to T). While the model is different to [9], in outline the security arguments are similar.

In 'Two Party Adaptor Signatures' by Erwig et. al. [10], the model of [9] is extended, considering how the same model can apply to any signature scheme derived from an ID protocol. It makes the insightful note that such signature adaptors are by definition impossible in unique signature schemes such as BLS [11]. This paper also formally extends the model to a multiparty signing context, which we argue in this paper is the only one of interest, but uses a KOSK ("knowledge of secret key") model, different from that in MuSig and MuSig2, referenced above.

4 Single key signature adaptor for BIP340 Schnorr (useless)

Given the definitions in BIP340 for public key P, message m, private key (here x) and signature (R, σ) we define a "signature adaptor" creation algorithm **AdaptorCreate** which takes as input a scalar $t \in \mathbb{Z}_N$, a key P, a message m, and a signature (R, σ) as defined above, and outputs a "signature adaptor", which is a tuple: (R, σ', T).

T will sometimes be referred to as the "adaptor point", and t as the corresponding "adaptor secret".

The algorithm to create the signature adaptor here is to calculate:

$$\begin{aligned} T &:= tG \\ \sigma' &:= \sigma - t \end{aligned}$$

...and then return (R, σ', T). The awkwardness of the inclusion of R in this return tuple is directly related to why this procedure is "useless", as we will see.

We then define an algorithm **AdaptorVerify** which takes as input a signature adaptor (R, σ', T) and calculates:

$$\sigma' G \overset{?}{=} R - T + \mathbb{H}(R||P||m)P$$

...returning true or false.

If the above definition looks wrong, please refer to the last paragraph of Section 5.

4.1 Claim 1

The above-defined single-key signature adaptor is honest-verifier zero knowledge

First we define an alternative algorithm for creating signature adaptors, which we call **AdaptorForge**:

1. Choose $k \in \mathbb{Z}_N$ at random
2. Set $R := kG$

3. Choose $\sigma' \in \mathbb{Z}_N$ at random
4. Set $T := R + \mathbb{H}(R||P||m)P - \sigma' G$

Hence, an adversary can produce a signature adaptor (R, σ', T), which can trivially be seen to pass **AdaptorVerify**, for any arbitrary values of R, σ' that you choose (even without the pre-existence of a genuine (R, σ) which would of course make it even more trivial). Note that, however, the adversary cannot find the value of t for the chosen signature adaptor, without the ability to extract EC discrete logs (proof: assume by contradiction that the adversarsy *can* find t, then, this would imply their ability to extract the private key x from P, because $x = \frac{t+\sigma'-k}{\mathbb{H}(R||P||m)}$ and the adversary has all of the RHS. Thus this adversary has extracted the DL of P without any interaction with an oracle etc.).

Treating execution of **AdaptorCreate** as an interaction with a challenger C who replies to calls to $\mathbb{H}$ with random values (the qualifier "honest-verifier" here occurs since we tacitly assume that the challenge values are genuinely random):

- Send R, P, m to C
- Receive $\mathbb{H}(R||P||m)$ from C
- Output T, σ', R

. . . we can follow the same logic as is applied to the Schnorr identification protocol, as in Boneh and Shoup [7] Thm 19.4: if the adversary can forge transcripts using **AdaptorForge** with the same statistical distribution as that seen from honest application of **AdaptorCreate**, then we have the HVZK property claimed.

AdaptorForge can function as *Sim* in the above HVZK definition, possessing only the public, not private key - it can choose σ' uniformly at random, and this defines T, which will also be uniformly random, assuming an idealized hash function. In the **AdaptorCreate** case, the value of t is chosen by the (private-key knowing) creator, and σ' is calculated as $\sigma - t$, where σ is calculated as a normal Schnorr signature and, as there, the output of the idealized hash function we posit, is uniformly random. In both cases, because the challenge (as modelled as the output of the RO) is random and independent of the input R, P, m, the distribution of the output values (T, σ') will be uniformly random.

Note that **AdaptorForge** is not some "discovery" or surprising result about forgery, it is well known and intrinsic property of such "signature adaptors", but it is crucial to development of concepts here. And as noted above, it was not mentioned in [9].

We have observed here two things: signature adaptors do not leak information about private keys (here, x), and also, signature adaptors can be trivially forged, i.e. they are not digital signatures. (Note of course that the former does not *imply* the latter; see actual digital signature algorithms for a counterexample).

Our claim therefore is that there is no need for more complex arguments w.r.t. unforgeability of the underlying signature scheme (here, BIP340 Schnorr) under the scenario of the creation of adaptors; by the definition of zero knowledge, the adversary has received zero extra information from the provision of any number of adaptors, and so nothing has changed.

4.2 Why single key signature adaptors are useless

The above algorithm is described as "useless" because it doesn't provide either:

- A way to create enforcement that the publication of a signature on a pre-defined message will reveal a secret
- A way to embed secret data that is not already possible, without this algorithm

On the first point, we observe that Schnorr signatures are not *unique*; one can create other signatures for the same context of P, m by simply using a different nonce R. Thus the output of **AdaptorCreate** (R, σ', T) will not allow the enforcement of the revelation of t, if a new signature is created, e.g. authorizing a Bitcoin transaction (the "pre-defined messsage"), using a different value of R. This is why the remainder of the document is about the MuSig case, where R can be fixed (in aggregated form) before signing is completed cooperatively.

(Note that Dryja's "discrete log contracts" [8] applies similar reasoning, "fixing" R, but for a different purpose: to prevent public equivocation by signing two messages).

On the second point, it is trivial that secret data can be hidden in public data, if the sender and receiver of the secret are able to share, in advance, some secret key; see e.g. the one-time pad. This is not to trivialize the usefulness, for steganography, of public randomness (and, in the Bitcoin blockchain for example, the largest part of the data is public randomness in the form of signatures and keys); just to say that signature adaptors aren't an algorithm for this purpose; simple subtraction or xor have the same effect.

5 Definitions for the multiparty (MuSig) case

We will start by defining:

- A **signing context** c consisting of a coordinated group of participants p of size n, as an ordered list $p_1 \dots p_n$, and a message for signing, m.
- An algorithm **KeySetup**, taking as input a set of keys Π_i for the participants p, and outputting a set of keys P_i, one for each participant. This algorithm is as defined in the MuSig paper. Note that we will use the notation x_i for the private keys of P_i (which are therefore the delinearized versions of the private keys of the starting keys Π_i). P_{agg} is defined as $\sum_i P_i$.
- An algorithm **Sign1Check**, run by each participant p_i, which takes as input a set of hash values h_j from each other participant $p_j, j \neq i$, and outputs "true" if and only if the received values from each participant are valid outputs of the preagreed commitment hash function $\mathbb{H}_c$.
- An algorithm **Sign2Check**, run by each participant p_i, which takes as input a set of public keys $R_j, j \neq i$ from each other participant, along with R_i, and outputs either "reject" if $\mathbb{H}_c(R_j) \neq h_j$, or, if all hashes match, then outputs $R_{agg} = \sum_{k=1}^{n} R_k$. Note that each R_k has a corresponding private key of k_k.

- An algorithm **AdaptorCreateM**, run by any participant p_i, which takes as input, the output of **KeySetup**, P_{agg}, and **Sign2Check**, R_{agg}, and outputs an adaptor: (T_i, σ_i'). Note that this tuple no longer includes R, as it did in the section on single-key adaptors.

Note that **Sign1Check** and **Sign2Check** are running the 1st and 2nd steps of the 3-round MuSig algorithm; see the MuSig paper for the full description.

Note also that **AdaptorCreateM** by definition can only run after the successful completion of the first three algorithms.

We now define three separate verification algorithms:

- $\mathbb{V}_s$ takes as input (σ, R, P, m) and outputs "true" or "false". This is exactly the Schnorr signature verification algorithm (to be more precise, we can specify the BIP340 verification algorithm). Note that this is not defined in terms of a signing context as per above, since it has no knowledge about the constitution of the public key P used as input. For ease of reference, the algorithm is: $\sigma G \overset{?}{=} R + \mathbb{H}(R||P||m)P$.
- $\mathbb{V}_p$ takes, as input $(c, \sigma_j, j, R_j, R_{agg}, P_{agg})$, where P_{agg}, R_{agg} are as output from **KeySetup** and **Sign2Check** respectively. It outputs "true" if and only if $\sigma_j G = R_j + \mathbb{H}(R_{agg}||P_{agg}||m)P_j$ (note that m is included in c), else returns "false".
- $\mathbb{V}_a$ takes, as input $(c, j, (T_j, \sigma_j'), R_j, R_{agg}, P_j, P_{agg})$, where (P_{agg}, P_j), R_{agg} are as output from **KeySetup** and **Sign2Check** respectively. It outputs "true" if and only if $\sigma_j' G = R_j - T_j + \mathbb{H}(R_{agg}||P_{agg}||m)P_j$ (note that m is included in c), else returns "false".

The two non-vanilla Schnorr verifications $\mathbb{V}_p$, $\mathbb{V}_a$ have long input tuples, but this can be simplified by specifying that algorithms **KeySetup**, **Sign1Check** and **Sign2Check** must have been run in advance and completed successfully (as is clearly required for the following security claims). Then the definitions become:

- $\mathbb{V}_p$ takes, as input (σ_j, j), outputs accept/reject.
- $\mathbb{V}_a$ takes, as input $((T_j, \sigma_j'), j)$, outputs accept/reject.

The reader may be more familiar with the equivalent definition of "signature adaptor": $\sigma_j' = k_j + \mathbb{H}(R_{agg} + T_j||P_{agg}||m)P_j$, for point T_j. The subtraction version given here is equivalent but a little easier to reason about, since the "tweak" which T constitutes, now appears only *outside* a hash function. To see the equivalence, redefine, for participant j, $R_j' = R_j + T_j$ as the generating step before **Sign1Check**, so that $R_{agg} = \sum_k R_k + T_j$ and thus V_a passes since $\sigma_j G = R_j' - T_j + \mathbb{H}(R_{agg}||P_{agg}||m)P_j$.

6 Security Properties

6.1 Claim 2

A corollary to the above definitional choices:

*Assume that the first algorithms (**KeySetup**, **Sign1Check** and **Sign2Check**) have run and completed successfully. Assume further that a participant p_j has run the algorithm **AdaptorCreateM**, creating a signature adaptor (T_j, σ'_j) that passed $\mathbb{V}_a$. Then, there is an exact one-one mapping between $\sigma'_j \leftrightarrow T_j$, that is, there can be no second T^* such that $\mathbb{V}_a$ passes for (T^*, σ'_j). Thus the binding is perfect, not only computational.*

This can be seen from the following: R_{agg}, P_{agg}, R_j, m and P_j are defined at the end of **Sign2Check**. Thus in the equation $\sigma_j^{'} = k_j - t_j + \mathbb{H}(R_{agg}||P_{agg}||m)x_j$, all terms are fixed except for $\sigma_j^{'}$ and t_j. A choice of one (in the field $\mathbb{Z}_N$) defines the other.

This core property is of course crucial: if our goal is to create algorithms that enforce the revelation of a secret, we need the adversarial adaptor creator not to be able to equivocate and "reveal" a different secret.

A caution about the claim of "perfect" binding here: it is true *in the sense that for fixed context $(c, R_{agg}, P_{agg}, m, \sigma'_j)$ there is no ability to forge an alternative.* The real practical security limitation is that of MuSig itself, i.e. the ability or inability of an adversary to forge a multisignature without honest counterparties' consent.

6.2 Claim 3

*The HVZK property applies to **AdaptorCreateM** as it did to **AdaptorCreate***

Here we define an algorithm **AdaptorForgeM**:

- Assume the adversary is index j in c, and that **KeySetup**, **Sign1Check** and **Sign2Check** have all been completed successfully.
- Adversary then chooses $\sigma'_j \in \mathbb{Z}_N$, at random.
- Then calculates $\mathbb{H}(R_{agg}||P_{agg}||m)P_j - \sigma'_j G = Q$.
- Then calculates $T_j = Q + R_j$.
- Return (T_j, σ'_j)

As can be checked, (T_j, σ'_j) passes $\mathbb{V}_a$, though the creator of this tuple does not know the discrete log of T_j, nor the discrete log of P_j.
Since, therefore, the "forgeability" of signature adaptors is as trivial in the multi-participant aggregated key and nonce context, as it was in the single-key context, the argument for honest verifier zero knowledge carries through in the same way. Simply observe that the statistical distribution of outputs of **AdaptorForgeM** is the same as that of **AdaptorCreateM**.

6.3 Source of the "forgeries" AdaptorForge(M)

This section is an attempt to "bed in" to the reader's mind, the reasons why adaptors have the properties they have.
The basic Schnorr signature design relies on the fact that the application of the Fiat-Shamir heuristic *fixes all the elements of the conversation transcript that*

occur before the challenge. This prevents the adversary from back-solving the response element in the Σ-protocol without knowing the secret. This can be applied to the case of an adaptor, thus:
As applied to Claim 1, if the committed value is R, then we have not committed to the effective secret nonce value, used in the adaptor: $k - t$, but to an offset of it, k, for an undefined value t, hence we can back solve the value of T for an arbitrary σ'.
As applied to Claim 3, the committed value is R_{agg}, and the nonce R_j was committed to in step **Sign1Check**, so those values are fixed, but we have not committed to the effective secret nonce $k_j - t_j$, because T_j is an offset from R_j that can be back-solved after the event. Fixing the R *hashed into the challenge* didn't remove forgeability, because the forger has "wiggle room" in the nonce offset $k - t$.

6.4 Unforgeability of partial signatures in the presence of adaptors

In this section we discuss in detail how the use of adaptors in one or more signing sessions does or does not affect the unforgeability of partial (and, indeed, full) signatures as defined in MuSig.
As a prelude, we precis here the steps of the MuSig18 [10] signing protocol over a message m, for a set of N parties, indexed $i = 1 \ldots N$, inserting adaptors at the appropriate point:

1. **KeySetup**: Each party announces key Π_i, shares, all parties agree on $P_{agg} = \sum \mathbb{H}(L, \Pi_i)\Pi_i$ where L is the serialization of the keys $\Pi_1 \ldots \Pi_N$. A reminder that P_i in this document refers to $\mathbb{H}(L, \Pi_i)\Pi_i$.

2. **Sign1Check**: Share nonce commitments, each party sends $\mathbb{H}(R_i)$ to all others.

3. **Sign2Check**: Share nonces R_i and verify others' nonce commitments. Calculate $R_{agg} = \sum R_i$.

4. (not part of MuSig, but, here is where adaptors will be shared; see **AdaptorCreateM**). The details, including ordering, will be discussed for certain situations below.

5. **Round 3**: Share partial signatures $\sigma_i = k_i + \mathbb{H}(P_{agg}|R_{agg}|m)x_i$ (see Section 5 for the definition of x_i). By correctness, $\sum \sigma_i$ is a valid Schnorr signature on message m by the aggregate key P_{agg}.

We will refer to this numbered list, in some of the cases below:

6.4.1 Base case: single adaptor by single party in one signing session

Consider a case in which only one index j provides an adaptor σ'_j for a signing session. In this case, we make a further claim:

6.4.2 Claim 4

Assume that the first three steps have run and completed successfully. Assume further that a participant p_j has run the algorithm ***AdaptorCreateM****, producing (σ'_j, T_j). The partial signature σ_j cannot be forged after these steps, by any subset of the counterparties $i \neq j$ unless they can solve the Elliptic Curve Discrete Logarithm Problem.*

Proof: We show a typical reduction: that an adversary $\mathbb{A}$, controlling all parties $p_i, i \neq j$, that can calculate σ_j, can extract arbitrary discrete logs.

We wrap the adversary $\mathbb{A}$ and take as challenge, a point Q, for which we are going to extract the discrete log using $\mathbb{A}$. We run steps 1-3 above, then calculate an adaptor, using the following forgery procedure, which is similar to that of **AdaptorForge** in Claim 1, but with an important modification; we don't assume that we already know the secret nonce k:

1. The challenger gives us a point Q. We set $R_j := Q$ (note, without yet knowing its discrete log).
2. We, as party p_j, along with $\mathbb{A}$, as all other parties, follow steps 1-3 of the above, resulting in a P_{agg} and R_{agg} value as usual.
3. We *forge* an adaptor signature as follows (see Claim 1): first, choose σ'_j at random. Then calculate $T_j = R_j + \mathbb{H}(R_{agg}|P_{agg}|m)P_j - \sigma'_j G$. Send, to $\mathbb{A}$, the tuple (σ'_j, T_j), which verifies according to $\mathbb{V}_a$.
4. $\mathbb{A}$ returns, with some non-negligible probability, a valid partial signature for our index, σ_j.
5. We can now calculate $k_j = \sigma_j - \mathbb{H}(P_{agg}|R_{agg}|m)x_j$, which we return as the discrete log of Q.

Note the unusual aspect of this procedure: while we act as the "honest" party, p_j, following all steps of the protocol as requested, we do *not* know the discrete log of R_j initially, and so cannot actually make the partial signature ourselves. However the fact that the adversary $\mathbb{A}$ *can* do this for us, effects the required ECDLP reduction.

Also note that the structure of this argument is radically simpler than that to show a reduction to ECDLP, or OMDL, for MuSig; the reason for this is that we are not trying to prove soundness of that signature scheme; that proof (for MuSig) already exists. Here we are only trying to prove that for the *honest party* (p_j), who we assume already to know the secret key x_j, there is no risk of external forgery introduced by adding the adaptor; and what makes *that* fairly simple, is the forgeability property discussed at length previously.

6.4.3 Second case: Multiple adaptors,one signing session

Perhaps a less likely practical scenario, but it is possible for party p_j to issue, in Step 4, *more than one* adaptor for corresponding adaptor points $T_q, q \in 1 \dots m$, in the same signing session. Note an algebraic nuance: in such a scenario, it is not only the case that the revelation of the full partial signature σ_j will reveal all the corresponding secrets t_q at the same time, but also, if one such secret such as t_1 is revealed, then all the other secrets t_q, *and* the full partial σ_j are revealed. There is essentially "only one secret value". Still we cover this case as it could conceivably matter.

In this case, the argument from the previous section carries over more or less identically. We generate m random values $\sigma'_{j,q}$ and follow the same forgery process for each of them, while, as before, setting R_j (remember that there is still only one such value, here) to Q given by the challenger. A successful full partial signature (σ_j) forgery by $\mathbb{A}$ reveals the discrete log of the challenged point Q.

Caveat: due to application of the function $\mathbb{H}$ in the calculation of the values T_q in the forgery as per Claim 1, this argument only applies to a scenario in which the adaptor points T have random distribution.

6.4.4 Third case: One adaptor, multiple signing sessions

A more specifically interesting application of "multiple adaptors at the same time": suppose Alice wants to provide signature adaptors on a point T_A for m *different* signing contexts which she has set up with N different counterparties Bob, Carol etc. Indeed, the most basic application of adaptors would be the adaptor-based CoinSwap [14], which is the two party case of this setup.

This third case cannot be demonstrated secure against partial signature forgery in the same way as Case 1. The reason is that our algorithm **AdaptorForgeM** can only be applied while generating an unpredictable value T; if we need to use the *same* T in multiple signing contexts, it cannot be used directly.
We need to apply two additional constraints/assumptions the argument in order for the reduction to go through. First,we rely on the Random Oracle Model so as to allow us to patch the output of the hash function, as seen by the adversary, at certain points. Second, there is a restriction on sequencing: **Sign1Check** and **Sign2Check cannot be done in parallel across all the different signing sessions**.
Finally to note: more tightness in the reduction must be lost, principally because there are multiple forgeries possible, i.e. principally a linear factor of m.

We thus define $\mathbb{A}_2$ as: (1) it takes the role of all counterparties at indices $2 \dots N$ in the m signing sessions. (2) It is thus given inputs: valid outputs of **KeySetup**, **Sign1Check** and **Sign2Check**. Note that as per above, these steps are completed in total, for each of the m signing sessions, sequentially, and *not* in parallel. (3) It is *then* given $\sigma'_{1,1}, \sigma'_{1,2}, \dots \sigma'_{1,m}$ where $\sigma_{x,y}$ is the signature adaptor by the party at index x in the y-th signing session. (4) $\mathbb{A}_2$'s responsibility is then to output a valid partial signature forgery $\sigma_{1,q}$ for *at least one of the m signing sessions*, with non-negligible probability.
We now describe the reduction to ECDLP based on a more sophisticated version of the game shown in 6.4.2.

1. The challenger gives us a point Q. We set $R_{1,1} := Q$.
2. All parties can run **KeySetup** for MuSig to agree on $P_{agg,q}$ for all m signing sessions (indexed by q).

3. We, as party p_1, along with $\mathbb{A}_2$, as all other parties, in **only the first signing context**, run **Sign1Check** and **Sign2Check** resulting in $R_{agg,1}$ as usual.

4. We now *forge* a signature adaptor *for signing context 1* as in 6.4.2: first, choose $\sigma'_{1,1}$ at random. Then calculate $T_1 = R_{1,1} + \mathbb{H}(R_{agg,1}|P_{agg,1}|m_1)P_{1,1} - \sigma'_{1,1}G$. At this point, the value (shared across signing sessions) of T_1 is set.

5. Next, for all the remaining signing session $q \in 2 \ldots m$, we do the following: (1) Choose $\sigma'_{1,q}$ at random. (2) Choose a random value for H for $\mathbb{H}(R_{agg,q}|P_{agg,q}|m_q)$ (note that we do not yet know $R_{agg,q}$). (3) Calculate $R_{1,q} = T_1 + \sigma'_{1,q}G + HP_{1,q}$.

6. Now we run **Sign1Check** and **Sign2Check** for each index $q \neq 1$, having defined $R_{1,q}$. Note that these steps do *not* require queries to the hash function $\mathbb{H}$ but only to hash $\mathbb{H}_c$ which is used for commitments. At this point we have $R_{agg,q}$ and so we can patch $\mathbb{H}(R_{agg,q}|P_{agg,q}|m_q) := H$.

7. Having completed the above two steps for all indices $2 \ldots m$, we have a full set of signature adaptors $(\sigma'_{1,2}, T_1), \ldots, (\sigma'_{1,m}, T_1)$ which we pass to $\mathbb{A}_2$.

8. All of these adaptors will, for $\mathbb{A}_2$, pass $\mathbb{V}_a$ verification as can easily be checked, due to our random oracle patching as per above.

9. $\mathbb{A}_2$ returns, with some non-negligible probability, say p, a valid partial signature $\sigma_{1,q}$ **for at least one signing session** $q \in 1 \ldots m$.

10. With probability $\frac{p}{m}$, $\mathbb{A}_2$ return a partial signature for the first signing session. In that case, we calculate $k_{1,1} = \sigma_{1,1} - \mathbb{H}(P_{agg,1}|R_{agg,1}|m)x_{1,1}$, which we return as the discrete log of Q.

Some notes are in order:

- There is an unjustified assumption in step 10 above; we assume that the adversary $\mathbb{A}_2$ emits a forgery with equal probability for each of the signing indices. The intuition behind this assumption is that the *new* information presented to $\mathbb{A}_2$ (when compared with a vanilla MuSig signing session), is presented all at once: the set of signature adaptors. However that is not rigorous, so perhaps the argument can be tightened up a bit.

- The loss of tightness of the reduction is not going to be *exactly* a factor of m, in any case: the random oracle patching introduces another factor (due to the collision possibility), though we presume this is negligible and not important.

- there is no claim here that running the **Sign1Check** and **Sign2Check** steps of all of the different signing sessions in parallel is *insecure*; on that, we offer no opinion. But this particular reduction approach requires running **one** of them **before** all the others.

6.4.5 Fourth case: multiple parties with their own adaptors in multiple signing sessions

Each party in a (set of) signing session(s) needs the security property that their *own* partial signatures can not be forged by any set of counterparties. We believe the arguments above already address this. If the (adversarial) counterparties produce their own signature adaptors, it only provides extra information to the honest counterparty, and therefore cannot represent an extra security risk. This statment assumes that we are sticking to the sequencing of events 1-5 as shown at the start of Section 6.4.

7 References

1. "Adaptor signatures", Poelstra, 2017 `https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-03-mit-bitcoin-expo/slides.pdf`
2. MuSig (3-round), Maxwell, Poelstra, Seurin, Wuille `https://eprint.iacr.org/2018/068`
3. MuSig2, Nick, Ruffing, Seurin 2021: `https://eprint.iacr.org/2020/1261`
4. Bitcoin Improvement Proposal 340 (Schnorr signatures) `https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki`
5. Bitcoin Improvement Proposal 327 (MuSig2 protocol) `https://github.com/bitcoin/bips/blob/master/bip-0327.mediawiki`
6. Multiparty S6 (blog; how to do a multiparty adaptor-based swap). `https://reyify.com/blog/multiparty-s6`
7. Boneh and Shoup's Graduate Course in Cryptography (contains a lot of useful fundamentals on security proofs) `https://toc.cryptobook.us/`
8. Discreet Log Contracts, Dryja `https://adiabat.github.io/dlc.pdf`
9. Generalized Channels from Limited Blockchain Scripts and Adaptor Signatures, Aumayr et al `https://eprint.iacr.org/2020/476`
10. Two-Party Adaptor Signatures From Identity Schemes, Erwig, Faust, Hostakova, Maitra, Riahi `https://eprint.iacr.org/2021/150`
11. Short signatures from the Weil pairing, Boneh, Lynn, Shacham 2001 `https://www.iacr.org/archive/asiacrypt2001/22480516.pdf`
12. Anonymous Atomic Locks for scalability in Payment Channel Hubs, Tairi, Moreno-Sanchez, Maffei 2019 `https://eprint.iacr.org/2019/589`
13. One-Time Verifiably Encrypted Signatures, Fournier 2019 `https://eprint.iacr.org/2019/589`
14. Scriptless script based atomic swaps, Blockstream Research `https://github.com/BlockstreamResearch/scriptless-scripts/blob/master/md/atomic-swap.md`
15. Adaptor signatures, summary by Bitcoin OpTech `https://bitcoinops.org/en/topics/adaptor-signatures/`