Block-level Accountability for Transparent Sanitizable Signatures

Kai Samelin, Henrich C. Pöhls,
Joachim Posegga and Hermann de Meer
{ks,hp,jp}@sec.uni-passau.de, demeer@uni-passau.de

Institute of IT-Security and Security-Law (ISL), University of Passau, Germany

Technical Report, Number MIP-1209
Department of Informatics and Mathematics
University of Passau, Germany
December 2012

Block-level Accountability for Transparent Sanitizable Signatures

Kai Samelin[1]*, Henrich C. Pöhls[2]**, Joachim Posegga[2], Hermann de Meer[1]

[1] Chair of Computer Networks and Computer Communication
[2] Chair of IT-Security
Institute of IT-Security and Security Law (ISL), University of Passau, Germany
{ks,hp,jp}@sec.uni-passau.de, demeer@uni-passau.de

Abstract. The paradigm of treating security properties on the block-level in sanitizable signature schemes was introduced by *Brzuska* et al. at EuroPKI'12. In this paper we extend their work in two respects:

First, we provide a new construction which retains transparency, a stronger privacy property. In particular, we formalize the property of block-level accountability for sanitizable signatures with transparency. The original work by *Brzuska* et al. did not allow for transparency. We derive a provably secure construction, achieving the new notion.

Second, a modification of our construction sacrifices transparency, but efficiently achieve *Brzuska* et al.'s stronger accountability notion also on the level of blocks. Our modified construction only requires a constant amount of signature generations to achieve block-level non-interactive public accountability property, which is a significant improvement over *Brzuska* et al.'s construction from EuroPKI'12.

We have implemented our constructions and the scheme introduced by *Brzuska* et al. at PKC'09 to provide a detailed performance analysis.

Keywords: Malleable Signatures, Accountability, Sanitizable Signatures

1 Introduction

Non-malleable signature schemes like RSA-PSS [3, 24] do not allow any modifications of the protected string of Bits. This behavior ensures that a third party can verify the integrity and authenticity of the signed message. However, there are many scenarios where signed data must be altered by a third party in a controlled way. Consider a driver's license which is digitally signed by the issuing

*The research leading to these results was supported by "Regionale Wettbewerbsfähigkeit und Beschäftigung", Bayern, 2007-2013 (EFRE) as part of the SECBIT project (http://www.secbit.de) and the European Community's Seventh Framework Programme through the EINS Network of Excellence (grant agreement no. [288021]).

**Is funded by BMBF (FKZ:13N10966) and ANR as part of the ReSCUeIT project

state. To protect the privacy of the driver's license holder, the holder is allowed to anonymize its name, while the date of birth can still be verified by any other party, e.g., a bouncer. Moreover, the government cannot be involved every time a document needs to be altered. Hence, a third party must be able to alter data without interacting with the original signer. This constellation is known as the "digital document sanitization problem", as formulated by *Miyazaki* et al. [21].

Sanitizable Signature Schemes (SanSig), introduced by *Ateniese* et al. [2], explicitly allow for controlled modifications of a signed message. In particular, a SanSig allows that a signed message $m = (m[1], m[2], \ldots, m[\ell])$, can be changed to a different message $m' = (m[1]', m[2]', \ldots, m[\ell]')$. For each *block* $m[i]$, the signer has to decide during signature generation whether a sanitization by a semi-trusted third party, a.k.a. the sanitizer, is admissible in the future. Moreover, the sanitizer neither requires the private key of the signer nor to perform any protocol runs with the signer to actually sanitize the message. Hence, the sanitizer is able to derive a new verifying message-signature pair (m', σ') on its own behalf, which is still related to the original signer.

Motivation. There are many additional application scenarios, e.g., anonymizing medical details before giving the medical records to hospital's accountant [19] or secure routing protocols [2]. However, the security model for accountable sanitizable signatures introduced in [2], formalized and extended in [5], only allows to decide which party is accountable for the *complete* message-signature pair (m, σ). In particular, they formalized the notion of *block-level non-interactive public accountability*. A non-interactive publicly accountable SanSig allows that every third party is able to decide which party is accountable for a given message-signature pair (m, σ) without requiring any auxiliary information besides what is given from the signature. If the accountable party cannot be derived without the auxiliary information, the scheme is said to be *transparent* [2]. Recently, *Brzuska* et al. introduced the paradigm of treating properties on the block-level [8]. In particular, [8] derives the notion of *block-level* non-interactive public accountability, i.e., a third party can decide which party is accountable for *each block* $m[i]$, so it cannot offer transparency. The question, if it possible to derive the accountability of each block while keeping transparency has not been answered. This online or interactive accountability keeps transparency as it requires that the accountability can only be derived, if the signer cooperates. This is the standard definition of accountability in transparent sanitizable signature schemes as given by *Ateniese* et al.. [2] and formalized by *Brzuska* et al. [5]. However, *Brzuska* et al. [8] did not achieve block-level accountability for transparent schemes, i.e., a block-level interactive or online-accountability. This paper gives a positive answer and provides a block-level interactive accountable and transparent sanitizable signature scheme.

As an application scenario for transparent and block-level accountable sanitizable signatures consider the case where a CEO signs a contract using a SanSig. Assume that each sentence of the contract is considered a block. The secre-

tary is allowed to change parts of the contract, e.g., the CEO allows sanitizing sentences to fill in dates, add names and correct spelling mistakes. Outsiders, especially the contractual partners, must not become aware of any corrections or changes by the secretary. Hence, the used SanSig must be transparent. However, for each sentence altered, the secretary shall later internally be deemed accountable, while each sentence not corrected or checked is under the full accountability of the CEO. This allows to later decide if the secretary has subsequently changed a sentence or not. With current schemes, this detailed level of accountability for each sentence is not possible. The current accountability notions only allow distinguishing if the secretary made no change to the document at all or at least one change somewhere. Hence, if the secretary is found to be accountable one could not say for which block, i.e., which sentence.

State of the Art. The standard security properties of SanSigs have first been introduced by *Ateniese* et al. [2]. They have later been formalized and extended by *Brzuska* et al. [5]. Limiting sanitizers to certain values has also been discussed [9, 14, 17, 22]. Later, *Brzuska* et al. introduced the concept of unlinkability, a privacy notion which prohibits a third party from linking two messages [7]. Currently, the notion of unlinkability combined with transparency requires the costly utilization of group signatures [7]. We thus focus on the security properties presented in [5]. In particular, unlike *Canard* et al. [10, 11], the signer needs to define which blocks are admissible during the signature generation, while we focus on a setting of a single signer and a single sanitizer, as transparent SanSigs for more than one sanitizer currently also require the costly use of group signatures [7, 10]. However, our ideas remain generally applicable in the multi-sanitizer setting.

Another malleable signature scheme are redactable signatures, introduced by *Johnson* et al. [16] and in a slightly different way by *Steinfeld* et al. [27]. In these malleable signature schemes, blocks cannot be modified: they can only be removed, i.e., redacted. Exemplary, refer to the work done in [1, 4, 12, 19, 20, 23, 25, 26, 28]. Approaches to combine redactable signature schemes and SanSigs appeared in [15]. In this paper, we focus on SanSigs, as redactable signatures schemes allow public redactions and are designed to achieve slightly different goals.

Our Contribution and Outline. This paper gives a positive answer to the question from *Brzuska* et al. [8]: We efficiently construct a scheme offering block-level accountability and transparency build only from standard primitives, i.e., an UNF-CMA digital signature scheme and a tag-based chameleon hash. In particular, we formalize the notion of block-level accountability for transparent sanitizable signatures and give a provably secure construction based on standard signature schemes and tag-based chameleon hashes [5, 18]. An alteration of our construction allows to achieve block-level non-interactive public accountability

equal to [8] more efficiently, with only a constant amount of signatures. Moreover, we have implemented our constructions and the SanSig introduced by *Brzuska* et al. in [5]. We provide a detailed performance analysis of our implementations and allow a comparison of speeds.

The rest of the paper is structured as follows: the standard security model our work is based upon is stated in Sect. 2. The notion of block-level accountability is introduced in Sect. 3. Our constructions and the corresponding performance evaluation are presented in Sect. 4, while our work is concluded in Sect. 5. All formal proofs are relegated to App. A.

2 Preliminaries

We shortly revisit the utilized algorithms, nomenclature and notations. They are derived from [5]. For a message $m = (m[1], \ldots, m[\ell])$, we call $m[i]$ a *block*. "," denotes a uniquely reversible concatenation, while $\perp \notin \{0,1\}^*$ denotes a special symbol not being a string, e.g., to indicate an error or an exception. A secure SanSig consists at least of the following algorithms:

Definition 1 (Sanitizable Signature Scheme). *Any SanSig consists of seven PPT algorithms* (KGen_{sig}, KGen_{san}, Sign, Sanit, Verify, Proof, Judge)*, such that:*

Key Generation. *There are two key generation algorithms, one for the signer and one for the sanitizer. Both create a pair of keys consisting of a private key and the corresponding public key, based on the security parameter λ:*

$$(\mathrm{pk}_{sig}, \mathrm{sk}_{sig}) \leftarrow \mathsf{KGen}_{sig}(1^\lambda), \quad (\mathrm{pk}_{san}, \mathrm{sk}_{san}) \leftarrow \mathsf{KGen}_{san}(1^\lambda)$$

Signing. *The Sign algorithm takes as input the security parameter λ, a message $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$, the secret key sk_{sig} of the signer, the public key pk_{san} of the sanitizer, as well as a description* ADM *of the admissibly modifiable blocks.* ADM *contains the number of blocks in m, denoted by ℓ, and a list of the indices of the modifiable blocks. ℓ is included in* ADM *to inhibit all attacks that maliciously try to append or remove blocks at the end. Note, we assume that* ADM *can always be correctly reconstructed from σ. It outputs the message m and a signature σ (or $\perp$, indicating an error):*

$$(m, \sigma) \leftarrow \mathsf{Sign}(1^\lambda, m, \mathrm{sk}_{sig}, \mathrm{pk}_{san}, \mathrm{ADM})$$

Sanitizing. *Algorithm Sanit takes the security parameter λ, a message $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$, a modification instruction* MOD*, a signature σ, the public key pk_{sig} of the signer and the secret key sk_{san} of the sanitizer. It modifies the message m according to the modification instruction* MOD*. We model* MOD *to contain a list of pairs $(i, m[i]')$ indicating that block i shall be modified into the string $m[i]'$. Note,* MOD *can be empty or the string $m[i]'$ can*

be equal to $m[i]$. Sanit generates a new signature σ' for the modified message $m' = \text{MOD}(m)$. Sanit outputs m' and σ' (or $\perp$ in case of an error:

$$(m', \sigma') \leftarrow \mathsf{Sanit}(1^\lambda, m, \text{MOD}, \sigma, \text{pk}_{sig}, \text{sk}_{san})$$

Verification. *The Verify algorithm outputs a bit $d \in \{\texttt{true}, \texttt{false}\}$ indicating the correctness of a signature σ for a message m with respect to the security parameter λ, the public keys pk_{sig} and pk_{san}:*

$$d \leftarrow \mathsf{Verify}(1^\lambda, m, \sigma, \text{pk}_{sig}, \text{pk}_{san})$$

Proof. *The Proof algorithm takes as input the security parameter λ, the secret signing key sk_{sig}, a message $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$ and a signature σ as well a set of (polynomially many) additional message-signature pairs $\{(m_i, \sigma_i) \mid i \in \mathbb{N}^+\}$ and the public key pk_{san}. It outputs a string $\pi \in \{0,1\}^*$ (or $\perp$ in case of an error):*

$$\pi \leftarrow \mathsf{Proof}(1^\lambda, \text{sk}_{sig}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, \text{pk}_{san})$$

Judge. *Algorithm Judge takes as input a message $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$ and a valid signature σ, the public keys of the parties and a proof π. It outputs a decision $d \in \{\texttt{Sig}, \texttt{San}, \perp\}$, indicating whether the message-signature pair has been created by the signer or the sanitizer (or $\perp$ in case of an error):*

$$d \leftarrow \mathsf{Judge}(1^\lambda, m, \sigma, \text{pk}_{sig}, \text{pk}_{san}, \pi)$$

We require that the usual correctness properties hold. In particular, every genuinely signed or sanitized message is accepted as valid by Verify. Moreover, every genuinely created correct proof by the signer makes Judge decide in favor of the signer. See [5] for a formal definition of these correctness requirements.

Note, again: MOD does not necessarily contain the instruction to modify the actual bits of a block: If $(i, m[i]') \in \text{MOD}$, then it might still be that $m[i]' = m[i]$. This allows a sanitizer to take accountability for a block $m[i]$ without modifying the contents of it.

Ateniese et al. introduced a set of desirable properties [2], later rigorously formalized by *Brzuska* et al. [5, 8]. We list the informal description of all of them for the paper to be self-contained:

- *Immutability* prevents the sanitizer from modifying blocks not admissible [5].
- *Unforgeability* assures that third parties cannot produce a signature for a "fresh" message. Fresh means the message has not been signed by the signer, nor issued by the sanitizer using the Sanit algorithm). This is similar to the unforgeability requirements of standard signature schemes. [5]

– *Privacy*, prevents third parties from recovering any original information from sanitized message parts. Its extension *unlinkability* [7], describes the "the impossibility to use the signatures to identify sanitized message-signature pairs originating from the same source" [6].

  As an example, a third party cannot retrieve any additional information besides what is given from a sanitized message and the signature [5]. Even if it a third party has access to two potentially differently sanitized versions of the same document, it cannot find out that they stem from the same source and hence cannot deduce any information, e.g., like recovering blocks by merging in the un-sanitized blocks from the other document [6].

– *Transparency* prevents third parties to decide which party is accountable for a given message-signature pair (m, σ). This is important, if the existence of a sanitizer must be hidden. This is required, if sanitization leads to disadvantages of any party involved [5].

– *Accountability* makes the origin (signer or sanitizer) of a signature undeniable. Hence, it allows a judge to settle disputes over the origin of a message [5]. The judge may request additional information from the signer. *Brzuska* et al. show [5] that accountability comes in two flavors: Signer- and Sanitizer-Accountability.

– *Non-Interactive Public Accountability* allows that a third party can always decide which party is accountable for a given *message*-signature pair (m, σ) [8] without additional interaction or auxiliary information.

– *Block-level Non-Interactive Public Accountability* allows that a third party can always decide which party is accountable for a given *block*-signature pair $(m[i], \sigma)$ [8] without additional interaction or auxiliary information-signature pair.

We restate the formal definitions of immutability, privacy, signer- and sanitizer-accountability, transparency, and the block-level public accountability to increase readability of Sect. 3, which introduces the new properties.

Definition 2 (Immutability). *A sanitizable signature scheme* SanSig *is immutable, if for any efficient algorithm* $\mathcal{A}$ *the probability that the experiment* $\mathsf{Immutability}_{\mathcal{A}}^{\mathsf{SanSig}}(\lambda)$ *given in Fig. 1 returns* 1 *is negligible (as a function of* λ*). The attacker gets access to a signing oracle. To break immutability, the adversary must either (1) be able to alter blocks not designated to be sanitized (directly or by manipulation of* ADM*), or (2) exchange the public sanitizer key to a new one which has not been input to the signing oracle.*

Definition 3 (Privacy). *A sanitizable signature scheme* SanSig *is private, if for any efficient algorithm* $\mathcal{A}$ *the probability that the experiment* $\mathsf{Privacy}_{\mathcal{A}}^{\mathsf{SanSig}}(\lambda)$ *given in Fig. 2 returns* 1 *is negligibly close to* $\frac{1}{2}$ *(as a function of* λ*). To breach*

Experiment $\mathsf{Immutability}^{\mathsf{SanSig}}_{\mathcal{A}}(\lambda)$
$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^\lambda)$
$(pk^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk_{\text{sig}},\cdots),\mathsf{Proof}(sk_{\text{sig}},\cdots)}(pk_{\text{san}})$
let (m'_i, σ'_i) for $i = 1, \ldots, q$ denote the answers from Sign indexed by i
return 1, if:
$\mathsf{Verify}(1^\lambda, m^*, \sigma^*, pk_{\text{sig}}, pk^*) = \texttt{true}$, and
$\forall i : pk^* \neq pk_{\text{san},i}$ or
$m^*[j_i] \neq m_i[j_i]$, where $j_i \notin \text{ADM}i$
//shorter messages are padded with $\perp$

Fig. 1. Immutability

Experiment $\mathsf{Privacy}^{\mathsf{SanSig}}_{\mathcal{A}}(\lambda)$
$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^\lambda)$
$(pk_{\text{san}}, pk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^\lambda)$
$b \leftarrow \{0, 1\}$
$a \leftarrow \mathcal{A}^{\mathsf{Sign}(sk_{\text{sig}},\cdots),\mathsf{Sanit}(\cdots,sk_{\text{san}}),\mathsf{Proof}(sk_{\text{sig}},\cdots),\mathsf{LoRSanit}(\ldots,sk_{\text{sig}},sk_{\text{san}},b)}(pk_{\text{sig}}, pk_{\text{san}})$
where oracle $\mathsf{LoRSanit}$ on input of:
$m_{0,i}, \text{MOD}_{0,i}, m_{1,i}, \text{MOD}_{1,i}, \text{ADM}i$
if $\text{MOD}_{0,i} \not\subseteq \text{ADM}i$, return $\perp$
if $\text{MOD}_{1,i} \not\subseteq \text{ADM}i$, return $\perp$
if $\text{MOD}_{0,i}(m_{0,i}) \neq \text{MOD}_{1,i}(m_{1,i})$, return $\perp$
let $(m_i, \sigma_i) \leftarrow \mathsf{Sign}(m_{b,i}, sk_{\text{sig}}, pk_{\text{san}}, \text{ADM}i)$
return $(m'_i, \sigma'_i) \leftarrow \mathsf{Sanit}(m_i, \text{MOD}_{b,i}, \sigma, pk_{\text{sig}}, sk_{\text{san}})$
return 1, if $a = b$

Fig. 2. Privacy

privacy, the adversary gives two crafted input messages to the *LoRSanit* *and has toto decide which one was used to produce the signed outcome.*

Definition 4 (Signer Accountability). *A sanitizable signature scheme* *SanSig* *is* signer accountable*, if for any efficient algorithm* $\mathcal{A}$ *the probability that the experiment* $\mathsf{Sig\text{-}Acc}^{\mathsf{SanSig}}_{\mathcal{A}}(\lambda)$ *given in Fig. 3 returns* 1 *is negligible (as a function of* λ*). In this game, the adversary has to generate a proof* π^* *which makes a genuine* *Judge* *decide that the sanitizer is accountable, even though it is not.*

Definition 5 (Sanitizer Accountability). *A sanitizable signature scheme* *SanSig* *is* sanitizer accountable*, if for any efficient algorithm* $\mathcal{A}$ *the probability that the experiment* $\mathsf{San\text{-}Acc}^{\mathsf{SanSig}}_{\mathcal{A}}(\lambda)$ *given in Fig. 4 returns* 1 *is negligible (as a function of* λ*). In this game, the adversary has to generate a valid signa-*

Experiment $\mathsf{Sig\text{-}Acc}^{\mathsf{SanSig}}_{\mathcal{A}}(\lambda)$
$(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^\lambda)$
$b \leftarrow \{0,1\}$
$(pk^*, \pi^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sanit}(\cdots, sk_{\text{san}})}(pk_{\text{san}})$
let (m'_i, σ'_i) for $i = 1, \ldots, q$
denote the answers from the oracle Sanit
return 1, if:
$\mathsf{Verify}(1^\lambda, m^*, \sigma^*, pk^*, pk_{\text{san}}) = \texttt{true}$, and
$(pk^*, m^*) \neq (pk_{\text{sig},i}, m'_i)$ for all $i = 1, \ldots, q$, or
$\mathsf{Judge}(1^\lambda, m^*, \sigma^*, pk^*, pk_{\text{san}}, \pi^*) = \texttt{San}$

Fig. 3. Signer Accountability

Experiment $\mathsf{San\text{-}Acc}^{\mathsf{SanSig}}_{\mathcal{A}}(\lambda)$
$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^\lambda)$
$b \leftarrow \{0,1\}$
$(pk^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdots, sk_{\text{sig}}, \cdots), \mathsf{Proof}(sk_{\text{sig}}, \cdots)}(pk_{\text{sig}})$
let $(m_i, \text{ADM}_i, pk_{\text{san},i})$ and σ_i for $i = 1, \ldots, q$
denote the queries to the oracle Sign
$\pi \leftarrow \mathsf{Proof}(1^\lambda, sk_{\text{sig}}, m^*, \sigma^*, \{(m_i, \sigma_i) \mid 0 < i \leq q\}, pk^*)$
return 1, if:
$\mathsf{Verify}(1^\lambda, m^*, \sigma^*, pk_{\text{sig}}, pk^*) = \texttt{true}$, and
$(pk^*_{\text{san}}, m^*) \neq (pk_{\text{san},i}, m_i)$ for all $i = 1, \ldots, q$, and
$\mathsf{Judge}(1^\lambda, m^*, \sigma^*, pk_{\text{sig}}, pk^*, \pi) = \texttt{Sig}$

Fig. 4. Sanitizer Accountability

ture σ^* which makes a genuine Proof generate a proof π, leading the Judge to decide that the signer is accountable, even though it is not.

Definition 6 (Transparency). *A sanitizable signature scheme SanSig is* proof-restricted transparent*, if for any efficient algorithm $\mathcal{A}$ the probability that the experiment $\mathsf{Transparency}^{\mathsf{SanSig}}_{\mathcal{A}}(\lambda)$ given in Fig. 5 returns 1 is negligibly close to $\frac{1}{2}$ (as a function of λ). The basic idea is that the adversary is not able to decide whether it sees a freshly signed signature or a signature created through Sanitize even though the attacker controls the input message*

Definitions allowing usage of deterministic signature schemes can be found in [5].

3 Block-level Accountability

In this section, we introduce the notion of block-level accountability:
First, we keep transparency, hence, the resulting construction (Sect. 4.2) preserves unforgeability, immutability, privacy and transparency, while it achieves

Experiment $\mathsf{Transparency}^{\mathsf{SanSig}}_{\mathcal{A}}(\lambda)$
$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^\lambda)$
$(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^\lambda)$
$b \leftarrow \{0,1\}$
$a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,sk_{\text{sig}},\cdot,\cdot),\mathsf{Sanit}(\ldots,sk_{\text{san}}),\mathsf{Proof}(sk_{\text{sig}},\ldots),\mathsf{Sanit/Sign}(\ldots,sk_{\text{sig}},sk_{\text{san}},b)}(pk_{\text{sig}}, pk_{\text{san}})$
where $\mathsf{Sanit/Sign}$ for input $m_i, \text{MOD}_i, \text{ADM}_i$
$\sigma_i \leftarrow \mathsf{Sign}(1^\lambda, m_i, sk_{\text{sig}}, pk_{\text{san}}, \text{ADM}_i)$,
$(m_i', \sigma_i') \leftarrow \mathsf{Sanit}(1^\lambda, m_i, \text{MOD}_i, \sigma_i, pk_{\text{sig}}, sk_{\text{san}})$
if $b = 1$:
$\sigma_i' \leftarrow \mathsf{Sign}(1^\lambda, m_i', sk_{\text{sig}}, pk_{\text{san}}, \text{ADM}_i)$,
finally return (m_i', σ_i').
return 1, if $a = b$ and $\mathcal{A}$ has not
queried any (m_i, σ_i) output by $\mathsf{Sanit/Sign}$ to Proof.

Fig. 5. Transparency

block-level accountability. Note, this does not achieve public non-interactive accountability, as we achieve transparency which is mutually exclusive to any public non-interactive accountability. We are the first to give a construction which allows block-by-block accountability, while fully achieving transparency.
Second, we re-state the stronger accountability notion of public non-interactive accountability, which our second construction (Sect. 4.3) fulfils. Henceforth, the second construction cannot preserve transparency, but achieves immutability, privacy and public non-interactive block-level accountability.

3.1 Block-level Interactive Accountability with Transparency

Let us give an informal definition of block-level accountability first:

> A SanSig offers *block-level accountability*, iff for all valid message-signature pairs (m, σ), with $m = (m[1], ...m[\ell])$, the algorithm Proof outputs a proof π which allows the algorithm $\mathsf{BlockJudge}$ to decide (i.e., $\mathsf{BlockJudge}$ outputs `Sig` or `San`), iff the given *block*-signature pair $(m[i], \sigma)$, with $m[i] \in m$, originates from the signer, or from the sanitizer, even in the presence of malicious signers/sanitizers.

This definition requires to introduce an additional algorithm $\mathsf{BlockJudge}$, which returns a list of d_i, where $d_i \in \{\texttt{Sig}, \texttt{San}\}$ indicates which party is accountable for the block $m[i]$. The additional algorithm $\mathsf{BlockJudge}$ is defined as follows:

$$d_i \leftarrow \mathsf{BlockJudge}(1^\lambda, m, \sigma, pk_{\text{sig}}, pk_{\text{san}}, \pi, i)$$

With this block-level notion, the standard message-level accountability notion for the complete message-signature pair (m, σ) becomes a corner case. To incorporate the standard accountability notion for the message level,

Experiment $\mathsf{Block\text{-}Signer\text{-}Acc}_{\mathcal{A}}^{\mathsf{SanSig}}(\lambda)$
$(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^{\lambda})$
$b \xleftarrow{\$} \{0,1\}$
$(pk^*, \pi^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sanit}(\cdots, sk_{\text{san}})}(pk_{\text{san}})$
Let $(m_j, \text{MOD}_j, \sigma_j, pk_{\text{sig},j})$ and (m'_j, σ'_j) for $j = 1, 2, \ldots, k$ be the queries and answers to and from oracle Sanit.
return 1, if:
$\mathsf{Verify}(1^{\lambda}, m^*, \sigma^*, pk^*, pk_{\text{san}}) = \texttt{true}$, and
$\forall j : pk_{\text{sig},j} \neq pk^*_{\text{sig}}$, or
$\exists q : \mathsf{BlockJudge}(1^{\lambda}, m^*, \sigma^*, pk^*, pk_{\text{san}}, \pi^*, q) = \mathsf{San}$, and
$(q, m'_j[q]) \notin \text{MOD}_j$

Fig. 6. Block-level Signer Accountability

Definition 7 (Message-level Accountability from Block-Level). *A sanitizer is accountable for a complete message-signature pair (m, σ), iff the SanSig is block-level accountable and there exists at least one block of m, for which BlockJudge identifies the sanitizer as accountable.*
Vice versa, a signer is accountable for the complete message-signature pair (m, σ), iff the SanSig is block-level accountable and BlockJudge identifies the signer to be accountable for all blocks of m.

This is the expected behavior, as defined by *Brzuska* et al. [5]. For block-level accountability, we now give rigorous definitions, that allow to formally include the existing definitions as a border case:

Definition 8 (Block-level Signer Accountability). *A sanitizable signature scheme SanSig is block-level signer accountable, if for any efficient algorithm $\mathcal{A}$ the probability that the experiment $\mathsf{Block\text{-}Signer\text{-}Acc}_{\mathcal{A}}^{\mathsf{SanSig}}(\lambda)$ given in Fig. 6 returns 1 is negligible (as a function of λ). Basically, the adversary wins if it can frame a genuine sanitizer identified by pk_{san}. To win the game the adversary has to generate a tuple $(\text{pk}^*, m^*, \sigma^*)$ including a proof π, which leads Judge to decide that a specific sanitizer identified by a fixed pk_{san} is accountable for a block q, while it is not.*

Definition 9 (Block-level Sanitizer Accountability). *A sanitizable signature scheme SanSig is block-level sanitizer accountable, if for any efficient algorithm $\mathcal{A}$ the probability that the experiment $\mathsf{Block\text{-}Sanitizer\text{-}Acc}_{\mathcal{A}}^{\mathsf{SanSig}}(\lambda)$ given in Fig. 7 returns 1 is negligible (as a function of λ). Basically, to win the game the adversary has to generate a tuple $(\text{pk}^*_{san}, m^*, \sigma^*)$ for which Proof generates a proof π which leads Judge to decide that a specific signer identified by a fixed pk_{sig} is accountable, while it is not. Note, with sanitizer accountability we as-*

Experiment $\mathsf{Block\text{-}Sanitizer\text{-}Acc}_{\mathcal{A}}^{\mathsf{SanSig}}(\lambda)$
$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^\lambda)$
$b \leftarrow \{0,1\}$
$(pk^*_{\text{san}}, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdots, sk_{\text{sig}}, \cdots), \mathsf{Proof}(sk_{\text{sig}}, \cdots)}(pk_{\text{sig}})$
Let $(m_i, \text{MOD}_i, \sigma_i, pk_{\text{san},i})$ and (m_i, σ_i) for $i = 1, 2, \ldots, k$ be the queries and answers to and from the oracle Sign.
$\pi \leftarrow \mathsf{Proof}(1^\lambda, sk_{\text{sig}}, m^*, \sigma^*, \{(m_i, \sigma_i) \mid 0 < i \leq q\}, pk^*_{\text{san}})$
return 1, if:
$\mathsf{Verify}(1^\lambda, m^*, \sigma^*, pk_{\text{sig}}, pk^*_{\text{san}}) = \texttt{true}$, and
$\forall i : pk_{\text{sig},i} \neq pk^*_{\text{sig}}$, or
$\exists q : \mathsf{BlockJudge}(1^\lambda, m^*, \sigma^*, pk_{\text{sig}}, pk^*_{\text{san}}, \pi, q) = \texttt{Sig}$ and
$(q, m'_j[q]) \in \text{MOD}_j$

Fig. 7. Block-level Sanitizer Accountability

sume the signer rebuttals an attack by generating genuine proofs which are given to Judge.

Theorem 1 (Block-level Signer Accountability $\implies$ Signer Accountability). *Every SanSig which is block-level signer accountable is also signer accountable.*

Proof. Assume towards contradiction, that there exists a block-level signer accountable scheme, which is not signer accountable on the message-level. Assume signer accountable on the message-level is build upon the block-level as described Then there exists a tuple (m^*, σ^*, π^*) for which the genuine algorithm Judge decides wrong, i.e., outputs `San` while the message is still untouched. This implies, that there exists one block $m^*[i]$ has not been generated by the sanitizer, which must be detected by BlockJudge to be block-level signer accountable. The contradiction follows.

Theorem 2 (Block-level Sanitizer Accountability $\implies$ Sanitizer Accountability). *Every SanSig which is block-level sanitizer accountable is also sanitizer accountable.*

Proof. Assume towards contradiction, that there exists a block-level sanitizer accountable scheme, which is not sanitizer accountable on the message-level. Then there exists a tuple (m^*, σ^*, π^*) for which the genuine algorithm Judge decides wrong, i.e., outputs `Sig` while the message has been touched by the sanitizer. This implies, that there exists one block $m^*[i]$ which has been generated by the sanitizer. Hence, this will be detected by BlockJudge as the scheme is block-level sanitizer accountable. The contradiction follows.

Definition 10 (Block-level Accountability). *A sanitizable signature scheme SanSig is block-level accountable, if it is block-level signer accountable and block-level sanitizer accountable.*

3.2 Block-level Non-interactive Public Accountability without Transparency

To simplify the notion of (block-level) public accountability, *Brzuska* et al. define that the algorithm Judge decides upon reception of empty proof $\pi = \perp$ [8]. In this paper, we stick with their approach for consistency. Obviously, transparency inhibits this public and instant form of accountability [8]. Following the aforementioned definitions, transparency and (block-level) public accountability are therefore mutually exclusive.

For the following formal definition of block-level non-interactive public accountability, we require the algorithm Detect [8]. It takes as input the security parameter, a message m and a valid signature σ together with the sanitizer's public key pk_{san} and the signer's public key pk_{sig}. Most notably, it also takes as an input a block index i and then returns `San` or `Sig`, indicating which party is accountable for the i^{th} block [8]:

Detect. On input of the security parameter 1^λ, a valid message-signature pair (m, σ), the corresponding public keys pk_{sig} and pk_{san}, and the block index i, Detect outputs the accountable party for block i (or $\perp$ in case of an error).

$$d \leftarrow \mathsf{Detect}(1^\lambda, m, \sigma, pk_{\text{sig}}, pk_{\text{san}}, i), d \in \{\texttt{San}, \texttt{Sig}, \perp\}$$

Definition 11 (Block-level Non-Interactive Public Accountability). *A sanitizable signature scheme SanSig together with an algorithm Detect is* block-level non-interactive publicly accountable*, if for any efficient algorithm $\mathcal{A}$ the probability that the experiment $\mathsf{Block\text{-}Pub\text{-}Acc}_{\mathcal{A}}^{\mathsf{SanSig}}(\lambda)$ given in Fig. 8 returns* 1 *is negligible (as a function of λ).*

4 Constructions

In this section, we derive two new constructions. The first new construction achieves block-level accountability with transparency. The second construction allows a more efficient block-level non-interactive public accountability requiring only a constant number of signatures. Note, for readability, we drop the security parameter λ for the rest of the paper.

4.1 Prerequisites

All implemented constructions make use of the tag-based chameleon hash by *Brzuska* et al. [5]. We shortly restate the requirements and their construction from [5]. In particular, the chameleon hash must be collision-resistant under random tagging-attacks as assumed and shown in [5].

Experiment $\mathsf{Block\text{-}Pub\text{-}Acc}^{\mathsf{SanSig}}_{\mathcal{A}}(\lambda)$
$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^\lambda)$
$(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^\lambda)$
$(pk^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\text{sig}}, \cdot, \cdot), \mathsf{Sanit}(\ldots, sk_{\text{san}})}(pk_{\text{san}}, pk_{\text{sig}})$
Let $(m_i, \text{ADM}_i, pk_{\text{san},i})$ and σ_i for $i = 1, 2, \ldots, k$
be the queries and answers to and from oracle Sign.
Let $(m_j, \text{MOD}_j, \sigma_j, pk_{\text{sig},j})$ and (m'_j, σ'_j) for $j = 1, 2, \ldots, k$
be the queries and answers to and from oracle Sanit.
return 1 if
$\mathsf{Verify}(1^\lambda, m^*, \sigma^*, pk_{\text{sig}}, pk^*) = \texttt{true}$, and
for m_i with $pk_{\text{san},i} = pk^*$, $\exists q$, such that
$\mathsf{Detect}(1^\lambda, m^*, \sigma^*, pk_{\text{sig}}, pk^*, q) = \texttt{Sig}$
and $m_i[q] \subseteq \text{MOD}_i$
return 1, if
$\mathsf{Verify}(1^\lambda, m^*, \sigma^*, pk^*, pk_{\text{san}}) = \texttt{true}$, and
for m_j with $pk_{\text{sig},j} = pk^*$, $\exists q$ such that
$\mathsf{Detect}(1^\lambda, m^*, \sigma^*, pk^*_{\text{sig}}, pk_{\text{san}}, q) = \texttt{San}$
and $m_j[q] \notin \text{MOD}_j$
return 0

Fig. 8. Block-level public accountability

Experiment $\mathsf{Rand\text{-}Tag}^{\mathcal{CH}}_{\mathcal{A}}(\lambda)$
$(pk, sk) \leftarrow \mathsf{CHKeyGen}(1^\lambda)$
$(\mathsf{TAG}, m, r, \mathsf{TAG}', m', r') \leftarrow \mathcal{A}^{\mathsf{OAdapt}(sk, \cdots)}(pk)$
where oracle OAdapt for the i^{th} query $(\mathsf{TAG}_i, m_i, r_i, m'_i)$ with $\mathsf{TAG}_i \in \{0,1\}^\lambda$
let $\mathsf{TAG}_i \leftarrow \{0,1\}^\lambda$ and compute $r'_i \leftarrow \mathsf{CHAdapt}(sk, \mathsf{TAG}_i, m_i, r_i, \mathsf{TAG}'_i, m'_i)$
return (TAG'_i, r'_i)
return 1, if
$(\mathsf{TAG}, m) \neq (\mathsf{TAG}', m')$ and
let $i = 1, \ldots, q$ denote the i^{th} oracle query
$\mathsf{CHash}(pk, \mathsf{TAG}, m, r) = \mathsf{CHash}(pk, \mathsf{TAG}', m', r')$ and
$\{(\mathsf{TAG}, m), \mathsf{TAG}', m')\} \neq \{(\mathsf{TAG}_i, m_i), \mathsf{TAG}'_i, m'_i)\}$ for all $i = 1, \ldots, q$ and
$\{(\mathsf{TAG}, m), \mathsf{TAG}', m')\} \neq \{(\mathsf{TAG}_i, m_i), \mathsf{TAG}'_j, m'_j)\}$ for all $i, j = 1, \ldots, q$

Fig. 9. Collision-Resistance against Random Tagging Attacks

Definition 12 (Collision-Resistance under Random-Tagging Attacks). *A tag-based chameleon hash $\mathcal{CH}$ is said to be collision-resistant under random-tagging attacks, if the probability that the experiment depicted in Fig. 9 returns* 1 *is negligible (as a function of λ).*

The following tag-based chameleon hash achieves this notion [5]:

Construction 1 (Chameleon Hash with Tags) *A chameleon hash $\mathcal{CH} :=$ (CHKeyGen, CHash, CHAdapt) with tags consists of three efficient algorithms:*

CHKeyGen. *The algorithm* $\mathsf{CHKeyGen}$ *takes as input the security parameter* 1^λ *and outputs the key pair required for the chameleon hash:*

1. *Generate RSA-parameter* n, e, d*, where* $e \in \{e_i \in \mathbb{N} \mid 2 < e_i < n, \gcd(\varphi(n), e_i) = 1\}$*, based on the security parameter* λ
2. *Choose a function* $\mathcal{H} : \{0,1\}^* \to (\mathbb{Z}/n\mathbb{Z})^\times$*, modeled as a random oracle*
3. *Output* $(\mathrm{sk}, \mathrm{pk})$*, where* $\mathrm{sk} = (d)$ *and* $\mathrm{pk} = (n, e, \mathcal{H})$

CHash. *The algorithm* CHash *takes as input the public key* pk*, a string* m *to hash, a tag* TAG *and a randomness* $r \in \{0,1\}^\lambda$*. It outputs:*

$$\mathcal{H}(\mathsf{TAG}, m) \cdot r^e \pmod{n}$$

CHAdapt. *The algorithm* $\mathsf{CHAdapt}$ *takes as input the private key* sk*,* m*,* m'*,* TAG*,* TAG'*,* r*. It outputs* $r' \leftarrow ((\mathcal{H}(\mathsf{TAG}, m) \cdot r^e) \cdot (\mathcal{H}(\mathsf{TAG}', m')^{-1}))^d \pmod{n}$

The formal proofs of correctness and security can be found in [5].

4.2 Block-level Accountable and Transparent SanSig

Next, we introduce a provably secure construction which is transparent, private, immutable, block-level accountable and unforgeable.

Construction 2 (Transparent and Block-level Accountable SanSig.) *Our construction is based upon the scheme introduced by Brzuska et al. [5]. In particular, each block is hashed using a tag-based chameleon hash. However, instead of using one tag for the complete message* m*, we use different tags for each block* $m[i]$*. We utilize a standard UNF-CMA signature scheme* $\mathcal{SS} = (\mathsf{SKeyGen}, \mathsf{SSign}, \mathsf{SVerify})$ *to generate the final signature. Moreover, like the original construction, we utilize a pseudorandom function* $\mathcal{PRF}$ *mapping* n*-bit input on a* n*-bit output for* n*-bit keys and a pseudorandom generator* $\mathcal{PRG}$ *mapping* n*-bit inputs to* $2n$*-bit outputs.*

$\mathsf{KGen}_{\mathbf{sig}}$. *Generate a key pair of the underlying signature algorithm* $\mathsf{SKeyGen}$*, i.e.,* $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{SKeyGen}(1^\lambda)$*. Pick a key* $\kappa \leftarrow \{0,1\}^\lambda$ *for the* $\mathcal{PRF}$*. Output* $(\mathrm{pk}_{sig}, \mathrm{sk}_{sig}) = (\mathrm{pk}, (\mathrm{sk}, \kappa))$*.*

$\mathsf{KGen}_{\mathbf{san}}$. *Generate a key pair of the underlying chameleon hash. In particular, output* $(\mathrm{pk}_{san}, \mathrm{sk}_{san}) \leftarrow \mathsf{CHKeyGen}(1^\lambda)$*.*

Sign. *On input of the message* $m = (m[1], \ldots, m[\ell])$*,* $m[i] \in \{0,1\}^*$*,* pk_{san}*,* sk_{sig} *and* ADM*, draw* $\ell + 1$ *nonces* n_i*,* $n_i \leftarrow \{0,1\}^\lambda$*. Compute* $x_i \leftarrow \mathcal{PRF}(\kappa, n_i)$ *and* $\mathsf{TAG}_i \leftarrow \mathcal{PRG}(x_i)$ *for all* $i = 0, \ldots, \ell$*. Draw* $\ell + 1$ *additional nonces* r_i*,* $r_i \leftarrow \{0,1\}^\lambda$*. Afterwards, compute:*

$$h[i] \leftarrow \begin{cases} \mathsf{CHash}(\mathrm{pk}_{san}, \mathsf{TAG}_i, m[i], r_i) & \text{if } i \in \text{ADM} \\ m[i] & \text{else} \end{cases}$$

for all $i = 1, \ldots, \ell$. *Let* $h[0] = \mathsf{CHash}(\mathrm{pk}_{san}, \mathsf{TAG}_0, (\mathsf{TAG}_1, \ldots, \mathsf{TAG}_\ell), r_0)$. *Generate* $\sigma_c \leftarrow \mathsf{SSign}(\mathrm{sk}, (h[0], h[1], \ldots, h[\ell], \mathrm{pk}_{san}, \mathrm{ADM}))$. *Output* (m, σ), *where* $\sigma = (\sigma_c, (\mathsf{TAG}_i)_{0 \le i \le \ell}, (n_i)_{0 \le i \le \ell}, \mathrm{ADM}, (r_i)_{0 \le i \le \ell})$

Verify. *On input of* pk_{sig}, pk_{san}, m *and* $\sigma = (\sigma_c, (\mathsf{TAG}_i)_{0 \le i \le \ell}, (n_i)_{0 \le i \le \ell}, \mathrm{ADM}, (r_i)_{0 \le i \le \ell})$ *compute:*

$$h[i] \leftarrow \begin{cases} \mathsf{CHash}(\mathrm{pk}_{san}, \mathsf{TAG}_i, m[i], r_i) & \text{if } i \in \mathrm{ADM} \\ m[i] & \text{else} \end{cases}$$

and $h[0] = \mathsf{CHash}(\mathrm{pk}_{san}, \mathsf{TAG}_0, (\mathsf{TAG}_1, \ldots, \mathsf{TAG}_\ell), r_0)$. *Output* $\mathsf{SVerify}(\mathrm{pk}, (h[0], h[1], \ldots, h[\ell], \mathrm{pk}_{san}, \mathrm{ADM}), \sigma_c)$

Sanit. *On input of* pk_{sig}, sk_{san}, m, MOD *and* σ *first check if the received message-signature pair is valid using* Verify. *Check, if* $\mathrm{MOD} \subseteq \mathrm{ADM}$. *If not, stop and output* $\perp$. *For each* ***block*** $(i, m[i]') \in \mathrm{MOD}$, *draw new nonces* $n_i' \leftarrow \{0,1\}^\lambda$ *and new tags* $\mathsf{TAG}_i' \leftarrow \{0,1\}^{2\lambda}$. *If* $i \notin \mathrm{MOD}$, *the tags, randoms and nonces are copied from the original signature, i.e.,* $n_i' = n_i$ *and* $\mathsf{TAG}_i' = \mathsf{TAG}_i$. *Always draw an additional nonce:* $n_0' \leftarrow \{0,1\}^\lambda$ *and an additional tag:* $\mathsf{TAG}_0' \leftarrow \{0,1\}^{2\lambda}$. *Compute:* $r_i' \leftarrow \mathsf{CHAdapt}(\mathrm{sk}_{san}, \mathsf{TAG}_i, m[i], r_i, \mathsf{TAG}_i', m[i]')$ *for each* $i \in \mathrm{MOD}$ *and* $r_0' \leftarrow \mathsf{CHAdapt}(\mathrm{sk}_{san}, \mathsf{TAG}_0, (\mathsf{TAG}_1, \ldots, \mathsf{TAG}_\ell), r_i, \mathsf{TAG}_0', (\mathsf{TAG}_1', \ldots, \mathsf{TAG}_\ell'))$. *Output* (m', σ'), *where* $m' \leftarrow \mathrm{MOD}(m)$ *and*

$$\sigma' = (\sigma_c, (\mathsf{TAG}')_{0 \le i \le \ell}, (n_i')_{0 \le i \le \ell}, \mathrm{ADM}, (r_i')_{0 \le i \le \ell})$$

Proof. *On input of* sk_{sig}, m, $\sigma = (\sigma_c, (\mathsf{TAG}_i)_{0 \le i \le \ell}, (n_i)_{0 \le i \le \ell}, \mathrm{ADM}, (r_i)_{0 \le i \le \ell})$, pk_{san} *and a sequence of message-signature pairs* $\{(m_i, \sigma_i) \mid i \in \mathbb{N}\}$, *search for a matching signature, such that for each block* $m[i]$, $(\mathrm{pk}_{san}, \mathsf{TAG}_i, m[i], r_i)$ *the following yields:*

$$\mathsf{CHash}(\mathrm{pk}_{san}, \mathsf{TAG}_i, m[i], r_i) = \mathsf{CHash}(\mathrm{pk}_{san}, \mathsf{TAG}_i', m'[i], r_i')$$

Set $TAG_i \leftarrow \mathcal{PRG}(x_i)$, *where* $x_i \leftarrow \mathcal{PRF}(\kappa, n_i)$. *Output* π, *where*

$$\pi = ((\mathsf{TAG}_i)_{0 \le i \le \ell}, m, \mathrm{pk}_{sig}, \mathrm{pk}_{san}, (r_i)_{0 \le i \le \ell}, (x_i)_{0 \le i \le \ell})$$

If any errors occur, it outputs $\perp$. *In other words,* Proof *outputs the originals of each block as the proof for the complete message.*

BlockJudge. *On input of* m, σ, pk_{sig}, pk_{san}, *an index* i, *and the proof* $\pi = ((\mathsf{TAG}_i^\pi)_{0 < i \le \ell}, m^\pi, \mathrm{pk}_{sig}^\pi, \mathrm{pk}_{san}^\pi, (r_i^\pi)_{0 < i \le \ell}, (x_i^\pi)_{0 \le i \le \ell})$, *first check, if* σ *verifies. Afterwards, check if* $\mathrm{pk}_{san}^\pi = \mathrm{pk}_{san}$. *Else, it return* $\perp$. *For block* $m[i]$ *in* m *let:*

$$d_i \leftarrow \begin{cases} \texttt{San} & \text{if } \mathsf{CHash}(\mathrm{pk}_{san}, \mathsf{TAG}_i, m[i], r_i) = \mathsf{CHash}(\mathrm{pk}_{san}^\pi, \mathsf{TAG}_i^\pi, m^\pi[i], r_i^\pi) \\ & \text{and the collision is non-trivial}, i \in \mathrm{ADM}, \mathsf{TAG}_i^\pi = \mathcal{PRG}(x_i^\pi) \\ \texttt{Sig} & \text{else} \end{cases}$$

Output $(d_i)_{0 < i \le \ell}$. *or* $\perp$ *on error.*

Judge. *The algorithm* Judge *gets* m, σ, pk_{sig}, pk_{san} *and the proof* $\pi = ((\mathsf{TAG}_i^\pi)_{0<i\leq\ell}, m^\pi, \mathrm{pk}_{sig}^\pi, \mathrm{pk}_{san}^\pi, (r_i^\pi)_{0<i\leq\ell}, (x_i^\pi)_{0<i\leq\ell})$ *as input. For each* $i \in$ ADM, *it calls* $d_i \leftarrow \mathsf{BlockJudge}(m, \sigma, \mathrm{pk}_{sig}, \mathrm{pk}_{san}, \pi, i)$. *On error, output* $\perp$. *If* $\exists i : d_i \neq \mathtt{Sig}$, *then output* `San` *and* `Sig` *otherwise.*

Theorem 3 (The Construction is Secure.). *If the underlying signature scheme* $\mathcal{SS}$ *is unforgeable, the used chameleon hash is collision resistant under random tagging attacks, while* $\mathcal{PRF}$ *and* $\mathcal{PRG}$ *are pseudorandom, our construction is transparent, private, immutable, block-level accountable and unforgeable.*

The proofs are relegated to App. A.

4.3 Block-level Non-Interactive Publicly Accountable SanSig

Next, we present a provably secure construction which is private, immutable, block-level non-interactive publicly accountable and unforgeable based on our first construction. This construction alters our first construction such that it removes transparency, but efficiently gives block-level non-interactive public accountability. We achieve this with a constant number of *signatures* compared to *Brzuska* et al.'s construction [8], where the number of signatures increases linearly with the number of blocks. However, we require linearly many hash operations.

Construction 3 (Block-level Non-Interactive Publicly Accountable SanSig.) *A secure construction which is private, immutable, block-level non-interactive publicly accountable and unforgeable, based on our first construction.*

$\mathsf{KGen}_{\mathbf{sig}}$. *Generate a key pair of the underlying signature algorithm* SKeyGen, *i.e.,* $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{SKeyGen}(1^\lambda)$. *Output* $(\mathrm{pk}_{sig}, \mathrm{sk}_{sig}) = (\mathrm{pk}, \mathrm{sk})$.

$\mathsf{KGen}_{\mathbf{san}}$. *Generate a key pair of the underlying chameleon hash and of an unforgeable signature scheme. In particular, let* $(\mathrm{pk}_c, \mathrm{sk}_c) \leftarrow \mathsf{CHKeyGen}(1^\lambda)$ *and* $(\mathrm{pk}_s, \mathrm{sk}_s) \leftarrow \mathsf{SKeyGen}(1^\lambda)$. *Output* $(\mathrm{pk}_{san}, \mathrm{sk}_{san}) = ((\mathrm{pk}_c, \mathrm{pk}_s), (\mathrm{sk}_c, \mathrm{sk}_s))$

Sign. *On input of the message* $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$, pk_{san}, sk_{sig} *and* ADM, *draw* ℓ *nonces:* $s_i = r_i \leftarrow \{0,1\}^\lambda$ *and* ℓ *additional nonces, i.e.,* $\mathsf{TAG}_i \leftarrow \{0,1\}^{2\lambda}$ *Compute:*

$$h[i] \leftarrow \begin{cases} \mathsf{CHash}(\mathrm{pk}_{san}.\mathrm{pk}_c, \mathsf{TAG}_i, m[i], r_i) & \text{if } i \in \text{ADM} \\ m[i] & \text{else} \end{cases}$$

for all $i = 1, \ldots, \ell$. *Generate* $\sigma_c \leftarrow \mathsf{SSign}(\mathrm{sk}_s, (h[1], \ldots, h[\ell], \mathrm{pk}_{san}, \text{ADM}, r_1, \ldots, r_\ell))$ *and* $\sigma_d \leftarrow \mathsf{SSign}(\mathrm{sk}_s, (h[1], \ldots, h[\ell], s_1, \ldots, s_\ell))$. *Output:*

$$\sigma = (\sigma_c, \sigma_d, (\mathsf{TAG}_i)_{0<i\leq\ell}, (r_i)_{0<i\leq\ell}, (s_i)_{0<i\leq\ell}, \text{ADM})$$

Verify. *On input of* pk_{sig}, pk_{san}, m, $\sigma = (\sigma_c, \sigma_d, (\mathsf{TAG}_i)_{0<i\leq\ell}, (r_i)_{0<i\leq\ell}, (s_i)_{0<i\leq\ell}, \mathrm{ADM})$, *compute:*

$$h[i] \leftarrow \begin{cases} \mathsf{CHash}(\mathrm{pk}_{san}.\mathrm{pk}_c, \mathsf{TAG}_i, m[i], s_i) & \text{if } i \in \mathrm{ADM} \\ m[i] & \text{else} \end{cases}$$

Check, if σ_d *either verifies under* $\mathrm{pk}_{san}.\mathrm{pk}_s$ *or* pk_{sig}. *If* σ_d *verifies under* pk_{sig}, *also check, if the* r_i *protected by* σ_c *and* σ_d *are equal, i.e., if* $r_i = s_i$. *If so, output:*

$$\mathsf{SVerify}(\mathrm{pk}, (h[1], \ldots, h[\ell], \mathrm{pk}_{san}, \mathrm{ADM}, s_1, \ldots, s_\ell), \sigma_c)$$

Sanit. *On input of* pk_{sig}, sk_{san}, m, MOD *and* $\sigma = (\sigma_c, \sigma_d, (\mathsf{TAG}_i)_{0<i\leq\ell}, (r_i)_{0<i\leq\ell}, (s_i)_{0<i\leq\ell}, \mathrm{ADM})$ *first check, if the received message-signature pair is valid using* Verify. *If not, stop outputting* $\perp$. *For each* ***block*** $(i, m[i]') \in \mathrm{MOD}$, *draw new tags* $\mathsf{TAG}_i' \leftarrow \{0,1\}^{2\lambda}$. *If* $(i, m[i]') \notin \mathrm{MOD}$, *set* $\mathsf{TAG}_i' = \mathsf{TAG}_i$ *and* $s_i' = r_i$. *Afterwards, compute:*

$$s_i' \leftarrow \mathsf{CHAdapt}(\mathrm{sk}_{san}.\mathrm{sk}_c, \mathsf{TAG}_i, m[i], r_i, \mathsf{TAG}_i', m[i]')$$

Output (m', σ'), *where* $m' \leftarrow \mathrm{MOD}(m)$ *and*

$$\sigma' = (\sigma_c, \sigma_d', (\mathsf{TAG}')_{0<i\leq\ell}, (r_i)_{0<i\leq\ell}, (s_i')_{0<i\leq\ell}, \mathrm{ADM})$$

where $\sigma_d' \leftarrow \mathsf{SSign}(\mathrm{sk}_{san}.\mathrm{sk}_s, (h[1], \ldots, h[\ell], s_1', \ldots, s_\ell'))$ *If* $i \notin \mathrm{MOD}$, *the tags and nonces are copied from the original signature. We want to emphasize, that* $r_i' = r_i$, *where* $i \in \mathrm{MOD}$, *is only possible with negligible probability. This can directly be derived from the definition of a chameleon-hash.*

Proof. *Always returns* $\perp$.

Detect. *On input of* $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$, *and* $\sigma = (\sigma_c, \sigma_d, (\mathsf{TAG}_i)_{0<i\leq\ell}, (r_i)_{0<i\leq\ell}, (s_i)_{0<i\leq\ell}, \mathrm{ADM})$, pk_{sig}, pk_{san}, $\perp$ *and an index* i, *first check, if* σ *verifies. For block* $m[i]$ *in* m *let:*

$$d_i \leftarrow \begin{cases} \mathtt{San} & \text{if } r_i \neq s_i \wedge i \in \mathrm{ADM} \\ \mathtt{Sig} & \text{else} \end{cases}$$

Output d_i *or* $\perp$ *on error.*

Judge. *The algorithm* Judge *gets* m, $\sigma = (\sigma_c, \sigma_d, (\mathsf{TAG}_i)_{0<i\leq\ell}, (r_i)_{0<i\leq\ell}, (s_i)_{0<i\leq\ell}, \mathrm{ADM})$, pk_{sig}, pk_{san} *and* $\perp$ *as input. First it checks, if* σ *verifies. For each* $i \in \mathrm{ADM}$, *it calls* $d_i \leftarrow \mathsf{BlockJudge}(m, \sigma, \mathrm{pk}_{sig}, \mathrm{pk}_{san}, \perp, i)$. *On error, output* $\perp$. *If* $\exists i : d_i \neq \mathtt{Sig}$, *output* `San` *and* `Sig` *otherwise.*

Theorem 4 (The Construction is Secure.). *If the underlying signature scheme* $\mathcal{SS}$ *is unforgeable, the used chameleon hash is collision resistant under random tagging attacks, while* $\mathcal{PRF}$ *and* $\mathcal{PRG}$ *are pseudorandom, our construction is private, immutable, block-level non-interactive publicly accountable and unforgeable.*

The proofs are relegated to App. A.

	Signing			Verifying			Sanitizing		
λ \ ℓ	100	500	1,000	100	500	1,000	100	500	1,000
512 Bit	16	63	125	15	46	78	157	766	1,641
1,024 Bit	28	112	14,132	20	96	22	1,007	4,948	9,720
2,048 Bit	110	391	750	62	328	657	7,109	35,328	70,997
4,096 Bit	563	1,546	2,798	250	1,235	2,469	54,719	272,672	545,062

Table 1. Performance of our first Scheme; Median Runtime in ms

	Signing			Verifying			Sanitizing			Detecting		
λ \ ℓ	100	500	1,000	100	500	1,000	100	500	1,000	100	500	1,000
512 Bit	16	78	140	15	47	94	172	797	1,578	16	46	94
1,024 Bit	47	172	313	31	141	265	1,047	5,062	10,438	32	125	266
2,048 Bit	172	516	969	94	437	875	7,547	36,079	72,735	93	421	859
4,096 Bit	922	2,157	4,141	328	1,546	3,546	55,453	271,329	562,683	360	1,546	3,109

Table 2. Performance of our second Scheme; Median Runtime in ms

4.4 Performance Measurements

We have implemented our SanSigs and the construction by *Brzuska* et al. [5]. The source code used for this evaluation will be made available on request. The tests were performed on a *Fujitsu Celsius* with an *Intel* Q9550 Quad Core @2.83 GHz and 3 GiB of RAM. We only used one core and utilized RSA as the signature algorithm. The moduli have been fixed to 512, 1,024, 2,048 and 4,096-Bit. We evaluated every algorithm with 100, 500 and 1,000 blocks. We fixed the amount of admissible blocks to 50% and always sanitized all admissible blocks. We omit the key pair generation, as we assume that the key pairs are pre-generated. Proof and Judge are very fast, as they contain only a database lookup, and are therefore omitted. The results can be seen in Tab. 1, Tab. 2, Tab. 3.

As seen, the performance is nearly the same for all three schemes. Hence, our constructions are as useable as the one by *Brzuska* et. al [5], while offering more possibilities and stronger security notions.

5 Conclusion and Open Questions

This paper answers the questions given by *Brzuska* et al. [8]. In particular, we have introduced and formalized the notion of block-level accountability. Based on these definitions, we have derived a provably secure construction relying on

	Signing			Verifying			Sanitizing		
λ \ ℓ	100	500	1,000	100	500	1,000	100	500	1,000
512 Bit	15	46	93	16	31	78	156	781	1,532
1,024 Bit	31	125	219	32	109	203	984	4,875	9,703
2,048 Bit	110	391	765	62	328	672	7,109	34,747	70,782
4,096 Bit	594	1,547	2,750	250	1,250	2,453	57,390	273,625	537,110

Table 3. Performance of the Scheme by *Brzuska* et al. [5]; Median Runtime in ms

the ideas of *Brzuska* et al. [5]. A small modification of our scheme also allows to achieve block-level public accountability with less signatures than the previously given constructions. The performance analysis of our implementations show that the scheme by *Brzuska* et al. [5] and our two new ones are reasonable performant, while our constructions achieve stronger security notions, as we have proven the block-level accountability notions imply the message level notions. It remains an open question how to offer the new paradigm of accountability to interlinked groups of blocks, i.e., accountability on a meta-block level. Furthermore, it is unclear how to mix public accountability and transparency in one message.

References

1. J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, A. Shelat, and B. Waters. Computing on authenticated data. Cryptology ePrint Archive, Report 2011/096, 2011. `http://eprint.iacr.org/`.
2. G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In *ESORICS: Proceedings of the 10th European Symposium on Research in Computer Security*, pages 159–177. Springer-Verlag, 2005.
3. M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *EUROCRYPT*, pages 92–111, 1994.
4. C. Brzuska, H. Busch, O. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder. Redactable Signatures for Tree-Structured Data: Definitions and Constructions. In *Proceedings of the 8th International Conference on Applied Cryptography and Network Security*, ACNS'10, pages 87–104. Springer, 2010.
5. C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of Sanitizable Signatures Revisited. In *Proc. of PKC 2009*, pages 317–336. Springer, 2009.
6. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Unlinkability of Sanitizable Signatures. In *Public Key Cryptography*, pages 444–461, 2010.
7. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schroeder. Unlinkability of sanitizable signatures. In *Public Key Cryptography*, pages 444–461, 2010.
8. C. Brzuska, H. C. Pöhls, and K. Samelin. Non-Interactive Public Accountability for Sanitizable Signatures. In *EuroPKI 9th European Workshop*, volume ???? of *LNCS*, pages ??–?? Springer-Verlag, 2012.
9. S. Canard and A. Jambert. On extended sanitizable signature schemes. In *CT-RSA*, pages 179–194, 2010.

10. S. Canard, A. Jambert, and R. Lescuyer. Sanitizable signatures with several signers and sanitizers. In *AFRICACRYPT*, pages 35–52, 2012.
11. S. Canard, F. Laguillaumie, and M. Milhau. Trapdoor sanitizable signatures and their application to content protection. In *ACNS*, pages 258–276, 2008.
12. E.-C. Chang, C. L. Lim, and J. Xu. Short Redactable Signatures Using Random Trees. In *Proceedings of the The Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology*, CT-RSA '09, pages 133–147, Berlin, Heidelberg, 2009. Springer-Verlag.
13. J. Gong, H. Qian, and Y. Zhou. Fully-secure and practical sanitizable signatures. In Xuejia Lai, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology*, volume 6584 of *Lecture Notes in Computer Science*, pages 300–317. Springer Berlin / Heidelberg, 2011.
14. S. Haber, Y. Hatano, Y. Honda, W. G. Horne, K. Miyazaki, T. Sander, S. Tezoku, and D. Yao. Efficient signature schemes supporting redaction, pseudonymization, and data deidentification. In *ASIACCS*, pages 353–362, 2008.
15. T. Izu, N. Kunihiro, K. Ohta, M. Sano, and M. Takenaka. Information security applications. chapter Sanitizable and Deletable Signature, pages 130–144. Springer-Verlag, Berlin, Heidelberg, 2009.
16. R. Johnson, D. Molnar, D. Song, and D.Wagner. Homomorphic signature schemes. In *Proceedings of the RSA Security Conference - Cryptographers Track*, pages 244–262. Springer, Feb. 2002.
17. M. Klonowski and A. Lauks. Extended Sanitizable Signatures. In *ICISC*, pages 343–355, 2006.
18. H. Krawczyk and T. Rabin. Chameleon Hashing and Signatures. In *Symposium on Network and Distributed Systems Security*, pages 143–154, 2000.
19. A. Kundu and E. Bertino. Structural Signatures for Tree Data Structures. In *Proc. of PVLDB 2008*, New Zealand, 2008. ACM.
20. K. Miyazaki, M. Iwamura, T. Matsumoto, R. Sasaki, H. Yoshiura, S. Tezuka, and H. Imai. Digitally Signed Document Sanitizing Scheme with Disclosure Condition Control. *IEICE Transactions*, 88-A(1):239–246, 2005.
21. K. Miyazaki, S. Susaki, M. Iwamura, T. Matsumoto, R. Sasaki, and H. Yoshiura. Digital documents sanitizing problem. Technical report, 2003.
22. H. C. Pöhls, K. Samelin, and J. Posegga. Sanitizable Signatures in XML Signature - Performance, Mixing Properties, and Revisiting the Property of Transparency. In *Applied Cryptography and Network Security, 9th International Conference*, volume 6715 of *LNCS*, pages 166–182. Springer-Verlag, 2011.
23. H. C. Pöhls, K. Samelin, J. Posegga, and H. de Meer. Length-hiding redactable signatures from one-way accumulators in $\mathcal{O}(n)$ (mip-1201). Technical report, University of Passau, 4 2012.
24. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26(1):96–99, 1983.
25. K. Samelin, H. C. Pöhls, A. Bilzhause, J. Posegga, and H. de Meer. On Structural Signatures for Tree Data Structures. In *Applied Cryptography and Network Security, 10th International Conference*, volume 7341 of *LNCS*, pages 171–187. Springer-Verlag, 2012.
26. K. Samelin, H. C. Pöhls, A. Bilzhause, J. Posegga, and H. de Meer. Redactable signatures for independent removal of structure and content. In *ISPEC*, volume 7232 of *LNCS*, pages 17–33. Springer-Verlag, 2012.
27. R. Steinfeld and L. Bull. Content extraction signatures. In *Information Security and Cryptology - ICISC 2001: 4th International Conference*. Springer Berlin / Heidelberg, 2002.

28. Z.-Y. Wu, C.-W. Hsueh, C.-Y. Tsai, F. Lai, H.-C. Lee, and Y. Chung. Redactable Signatures for Signed CDA Documents. *Journal of Medical Systems*, pages 1–14, December 2010.

A Proofs

Security of Construction 1. It is enough to show that the scheme is block-level accountable, transparent and immutable due to the implications given by *Brzuska* et al. [5, 8] and our implications. We prove each property on its own. Most of the proofs are kept short, as they are comparable to the ones given in [5].

Theorem 5 (Construction 1 is Secure.). *If the underlying signature scheme $\mathcal{SS}$ is unforgeable, while the used chameleon hash is collision-resistant under random-tagging attacks, our construction is transparent, private, immutable, unforgeable and block-level accountable. Following [5, 8], and our implications, it is enough to show that transparency, immutability and block-level accountability hold to prove the security of our scheme.*

Proof (Our scheme is immutable). Let $\mathcal{A}$ denote an efficient adversary breaking the immutability of our scheme. We can then construct an adversary $\mathcal{B}$ using $\mathcal{A}$ as a black box to break the unforgeability of the underlying signature scheme as follows. We simulate $\mathcal{A}$'s environment by simulating the signing oracle; the signature of the underlying signature scheme (σ_c) is generated by $\mathcal{B}$'s own oracle. Eventually, $\mathcal{A}$ will output a forgery attempt, i.e., a tuple $(pk^*_{\text{san}}, m^*, \sigma^*)$. This finishes the simulation. We have to distinguish between three cases: (1) We have $pk_{\text{san}} \neq pk_{\text{san},i}$ for all queries. As pk_{san} has been signed, the underlying signature scheme has been broken. (2) For some j and $i_j \notin \text{ADM}_j$, $m^*_j[i] \neq m_j[i]$ yields. As m^* has therefore not been queried, the unforgeability of the underlying signature scheme has been broken as well. (3) For some position i, the message has been replaced by a hash or vice versa resp. As this implies $\text{ADM} \neq \text{ADM}^*$, the signature must have been forged, as ADM is signed. If neither case happens, the simulation aborts. The signature forgeries can be extracted in all cases and are then returned by $\mathcal{B}$ as a valid forgery of the underlying signature scheme. Hence, $\mathcal{B}$'s success probability equals the one of $\mathcal{A}$.

Proof (Our scheme is transparent). Transparency follows from the uniform distributions of CHash and CHAdapt (random oracle), while the pseudorandom generators output numbers which are computationally indistinguishable from random. Transparency follows. We do not consider any tag-collisions here, as they only appear with negligible probability. Refer to [5] for a more formal proof of transparency.

Proof (Our scheme is block-level sanitizer accountable). Please note, that in the case where $h[i] \neq h^*[i]$, where h denotes the concatenation of blocks, implies a

direct forgery of the underlying signature scheme. This is also true for $pk_{\text{san},i} \neq pk^*_{\text{san}}$ and $\text{ADM}_i \neq \text{ADM}^*$. Also note, that in this case the Proof-oracle can trivially be simulated by picking κ itself. Hence, we can focus on the chameleon hash. To be successful, the adversary against block-level signer accountability needs to make sure that the proof algorithm Proof cannot find at least one non-trivial colliding pair of chameleon hash digests. Hence, we have:

$$\mathsf{CHash}(pk_{\text{san}}, \mathsf{TAG}_{j,0}, (\mathsf{TAG}_{j,i})_{0\leq i\leq \ell_j}, r_{j,0}) = \mathsf{CHash}(pk^*_{\text{san}}, \mathsf{TAG}^*_0, (\mathsf{TAG}_{j,i})^*_{0\leq i\leq \ell_j}, r^*_0)$$

for some query j. However, this is non-trivial and Proof can find the collision, a contradiction. This also applies for the outer chameleon hash, protecting against match-and-mix attacks. Building an extractor is straight forward and therefore omitted.

Proof (Our scheme is block-level signer accountable). Let $\mathcal{A}$ denote an efficient adversary breaking the block-level signer accountability of our scheme. We can then construct an adversary $\mathcal{B}$ using $\mathcal{A}$ as a black box to break the collision-resistance against random-tagging attacks of the underlying chameleon hash in the follow way. As before, $\mathcal{B}$ simulates $\mathcal{A}$'s environment. However, calls to the sanitization oracle are simulated using $\mathcal{B}$'s *OAdapt*-oracle and its own generated signature key pair. Eventually, $\mathcal{A}$ returns $(pk^*_{\text{sig}}, \pi^*, m^*, \sigma^*)$. As by definition, π^* must contain two (non-trivial) colliding tuples:

$$\mathsf{CHash}(pk_{\text{san}}, \mathsf{TAG}_{j,0}, (\mathsf{TAG}_{j,i})_{0<i\leq \ell}, r_{j,i}) = \mathsf{CHash}(pk^*_{\text{san}}, \mathsf{TAG}^*_i, (\mathsf{TAG}_{j,i})_{0\leq i\leq \ell})^*, r^*_i)$$

This finishes the simulation. Afterwards, $\mathcal{B}$ outputs the colliding tuples. These tuples break the collision-resistance of the chameleon hash as the tags are drawn at random. Any tag-collision is therefore only possible with negligible probability. Hence, $\mathcal{B}$'s success probability equals the one of $\mathcal{A}$. Please note that this also applies for the outer chameleon hash, protecting against match-and-mix attacks. Building an extractor is straight forward and therefore omitted. Note, the attack discovered by *Gong* et al. does not apply here, as we add an additional chameleon hash, protecting the whole message [13].

Security of Construction 2.

Theorem 6 (Construction 2 is Secure.). *If the underlying signature scheme $\mathcal{SS}$ is unforgeable, while the used chameleon hash is collision-resistant under random-tagging attacks, our construction is private, immutable, unforgeable and block-level non-interactive publicly accountable. Following [5, 8] and our implications, it is enough to show that privacy, immutability and block-level non-interactive public accountability hold to prove the security of our scheme.*

Proof. The proofs for privacy, immutability and unforgeability are exactly the same as for our first construction, with two notable exceptions: We do not achieve

transparency, while the "outer" signature protects against mix-and-match attacks. However, the sanitizer is able only to draw a new tag, which changes the random coin, but not the message, while the random coins for the chameleon hash are always distributed uniformly, which implies privacy. Therefore, we only need to show that our scheme is block-level non-interactive public accountability. Assume that there is an efficient adversary $\mathcal{A}$ against block-level non-interactive public accountability. We can then construct an adversary $\mathcal{B}$ using $\mathcal{A}$ as a black box to break the unforgeability of the underlying signature scheme as follows: $\mathcal{B}$ forwards any queries to its own oracles and returns the answers to $\mathcal{A}$. $\mathcal{B}$ also flips a coin $b \leftarrow \{0,1\}$. Eventually, $\mathcal{A}$ returns a tuple (pk^*, m^*, σ^*). If $b = 1$, $\mathcal{B}$ sets $pk^*_{\text{san}} \leftarrow pk^*$ and $(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}$ else, $\mathcal{A}$ sets $pk_{\text{sig}} \leftarrow pk^*$ and $(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}$. We therefore have to distinguish between two cases, i.e., a malicious sanitizer and a malicious signer. The probability that the simulation is done for the correct case is exactly $\frac{1}{2}$. We will omit cases where the random coins are equal, as this only occurs with negligible probability.

Malicious Signer. As $r'_i \neq r_i$, the underlying signature scheme must been forged, as σ_d protects all r_i, as $r'_i = r_i$ occurs only with negligible probability.

Malicious Sanitizer. We know that $r'_i = r_i$ only occurs with negligible probability. Therefore, σ_d must be a valid forgery.

In both cases, an extractor can trivially be build.