Black Box Accumulators in the Context of THEMIS*

Brave Research Team

February 8, 2021. Revision 1.0

1 Introduction

Black-Box Accumulators (BBA) were introduced in PETs 2016 by Jager and Rupp [1]. A BBA is a cryptographic token that lets users collect and sum up values (be they incentives, bonus points, reputation points, etc.) issued in a privacy-preserving manner, so that issuers are not able to later link different events of a given user. At the same time, malicious users are not able to cheat the system by pretending to have collected a higher amount than what was actually issued by authorized issuers.

As discussed in the RFC&C tech report, the goal of THEMISv2 is to decentralize the reward calculation of Brave Rewards protocol, while ensuring that Brave remains in control of which events are validated. On top of that, Brave should not be capable of linking different requests. We believe that BBAs are in many ways ideal for keeping the state of the interaction vector of THEMISv2, as they provide unlinkability, privacy, and integrity of the of the encoded vector.

The goal of BBAs in the context of THEMISv2 is to replace the token used in the current scheme (more information can be found here) with a more versatile token. The interaction between the user and the issuer are conducted as described in the RFC&C tech report: the user makes an initial request to receive a token that is used to keep the state of the user's ad interactions. At the end of the campaign, the user redeems the token for the earned BAT. The improvements we seek by introducing BBAs in the context of Brave Rewards are as follows:

- Tokens are linked to wallets during issuance and redemption; hence, we want to preserve the property that tokens can be redeemed only by the owner of the wallet to which they were issued;
- we want a token may contain additional attributes (such as, for example, the interaction vector);

*We assume that the reader is familiar with the RFC&C tech report

- we wish to facilitate a decentralized computation of the reward.

The construction over which we build this proposal is the one presented in BEKSS [2]. However, their proposal is constrained to a particular setting, which does not completely apply in the context of THEMISv2, and hence we modify certain properties of their construction. Mainly,

- We do not require partial spending. In our context, users redeem all points earned at once. Hence, we do not consider remainder token tracing.
- We require more than one counter. We want to keep state of each ad interaction, so that later, a decentralized reward computation can be performed.

The positive aspect of BEKSS construction is that the size of the token or the "show protocol" (when a recipient proves ownership of a token) does not grow with the number of attributes. Hence, in the context of THEMISv2, adding more counters (or different type of attributes) comes at no cost during confirmation events. Note that these events happen often, from devices which may have constrains in data usage and computational power (mobile devices); therefore, the size of the tokens and the complexity of the "show protocol" are crucial in the context of THEMISv2.

The goal of this report is to present an overview of what we use in THEMISv2. It falls out of the scope to present a formal description of the whole protocol or to provide a comprehensive security analysis. Our goal is to give sufficient material to allow one to perform a scalability study of the redemption phase, which is the most critical phase in the context of THEMISv2. In Section 2 we introduce the notation used throughout the document. Section 3 presents an overview of the solution we use, based on BEKSS construction. Then, we present the details of the redemption phase in Section 4, which is the main analysis goal for the scalability of the solution. Finally, in Section 5 we present some possible alternatives to BEKSS's construction.

2 Notation

Let λ be a security parameter. We write $a \xleftarrow{\$} A$ to denote that a is chosen uniformly at random from the set A. To denote vectors, we use bold letters, so $c_1, \ldots, c_N \in \mathbb{Z}_p^N$ is represented by $\boldsymbol{c}$. Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be cyclic groups (we use multiplicative notation) of prime order p. Let P and $\hat{P}$ be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. We call $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ a bilinear map or pairing if it is efficiently computable and the following holds:

- Bilinearity: $e(P^a, \hat{P}^b) = e(P, \hat{P})^{ab} = e(P^b, \hat{P}^a) \forall a, b \in \mathbb{Z}_p$.
- Non-degeneracy: $e(P, \hat{P}) \neq 1_{\mathbb{G}_T}$, i.e., $e(P, \hat{P})$ generates $\mathbb{G}_T$.

We adopt the Camenisch-Stadler notation [3] to denote zero-knowledge proofs and write, for example:

$$\Pi = SPK\{(x) : A = g^x \wedge B = A^x\}$$

to denote the non-interactive signature proof of knowledge that the prover knows the discrete *log* of A and B with bases g and A respectively, and that the discrete log is equal in both cases. We use $\Pi.\texttt{Verify}$ to denote the verification procedure of the proof. The input of the verifier is implicit in the proof definition. The `Verify` function outputs $\perp$ and $\top$ for failure or and success, respectively.

3 Adapting BEKSS's Construction

Several constructions of black box accumulators (also called, updatable anonymous credentials or privacy-preserving point collection) require the user to provide a zero-knowledge proof of ownership of a valid token/certificate. Some of the proposals based on the Algebraic MACs by Chase *et al.* [4], PS signatures, by Pointcheval and Sanders [5], or CL-signatures, by Camenisch and Lysyanskaya [6] fall under these requirements. However, the recent BEKSS paper uses a nice trick to avoid using ZKPs in the show procedure: they use structure-preserving signatures over equivalence classes [7] (SPS-EQ). This protocol takes a tuple (h, g) of group elements, and signs it. This signature can be adapted to all elements of the equivalence class, denoted by $[(h, g)]$, which consist of all exponentiations of the pair, mainly (h^c, g^c) for any $c \in \mathbb{Z}_p$. When adapting a signature to a different element of the equivalence class, the owner of the signature is making both instantiations unlinkable; i.e. it is randomising the tuple and the signature.

In the context of THEMISv2, the user holds an SPS-EQ signature, σ, over a vector, (C, P), which is a commitment of their state, i.e. the number of times they have interacted with each ad. For the structure of the commitment, the work follows the ideas of the algebraic MACs, PS-signatures or CL-signatures, of encoding the different counters in the exponent.

Each BBA has a single identifier, which is spent at the time of redemption. Moreover, it contains randomness chosen by the user in order to keep privacy of the requests. The user owns the committed state, the token identifier, and the randomness used in the token:

$$\tau = (C, P) = \left(h_1^{id} \cdot h_2^{r} \cdot \tilde{h}_1^{c_1} \cdots \tilde{h}_N^{c_N}, P\right)$$

where id is the identifier of the token, r is the randomization introduced by the user, and $c_1, \ldots, c_N$ are the different counters. The secret key of the server is a $N + 2$ tuple of scalars $sk = (sk_1, sk_2, sk_{c,1}, \ldots, sk_{c,N}) \in \mathbb{Z}_p^{N+2}$, such that $h_1 = P^{sk_1}, h_2 = P^{sk_2}$ and $\tilde{h}_i^{c_i} = P^{sk_{c,i}}$ for $i \in \{1, \ldots, N\}$. Moreover, the server owns a public-private SPS-EQ key-pair.

We now give an intuition of how the issuance and update protocols proceed. During the issuing phase, the token is signed with all counters initialised at zero. First, the user requests a signature over the tuple $\tau = (C^k, P^k) = \left((h_1^{id} \cdot h_2^r)^k, P^k\right)$, where $k \xleftarrow{\$} \mathbb{Z}_p$ is the randomness used during the issuance. The user provides a proof that the request is correct (the details of which fall out of the scope of the RFC&C). Then, the issuer verifies the proof, and produces an SPS-EQ over the pair, σ. Finally, at the time of reception, the user stores the token τ, the signature, σ, and the randomisation used during the request, $R = k$.

The SPS-EQ allows the user to make unlinkable confirmation events. More precisely, during the confirmation event, the user first randomises the token by computing $\tau' = (\tau_1^{k'}, \tau_2^{k'})$, where k' is chosen uniformly at random from $\mathbb{Z}_p^*$, and sends it together with the signature, σ', adapted to this new representation. Let the ad j be the one informed during the event. Upon receipt, the issuer parses $\tau' = (\tau_1', \tau_2')$ and verifies the validity of the signature, σ'. Then, it updates the token by letting $C_U = \tau_1' \cdot (\tau_2')^{sk_{c,j}}$ and producing an SPS-EQ over the new tuple (C_U, τ_2'). Upon receipt of the response, the user updates the stored randomisation, by multiplying it with the randomness used in the request, $R_{\texttt{New}} = R \cdot k'$. Finally, the user verifies that the update is correct with respect to the notified event.

The details of the issuance and update protocol are omitted in this report. What is most important is to understand how the redemption happens, and what the state of the token by that phase is, as is is this part that requires the scalability study. We dedicate a full section to that.

4 Redemption Phase

To fully understand the redemption phase it is necessary to understand the SPS-EQ signature scheme. Hence, we begin in Section 4.1 by giving a description of the signature scheme we used, as presented in [7]. In Section 4.2, we proceed by describing what are the steps that a user follows during redemption. Finally, Section 4.3 presents the steps followed by the verifier of the request.

4.1 Structure-Preserving Signatures over Equivalence Classes

We reproduce the formalisation of the signature scheme required in our construction (with the size of the vector equal to 2) for completeness. We refer the interested reader to the original publication [7] for more details. The signature scheme is described by 5 algorithms:

$\texttt{BGGen}(1^\lambda)$: On input a security parameter 1^λ, output a bilinear-group description $\texttt{BG} \xleftarrow{\$} \texttt{BGGen}(1^\lambda)$.

$\texttt{KeyGen}(\texttt{BG})$: On input a bilinear-group description, choose $(x_i)_{i\in[2]} \xleftarrow{\$} (\mathbb{Z}_p^*)^2$, set secret key $sk = (x_i)_{i\in[2]}$, compute public key $pk \leftarrow (\hat{X}_i)_{i\in[2]} = (\hat{P}^{x_i})_{i\in[2]}$ and output (sk, pk).

$\texttt{Sign}(M, sk)$: On input a representative $M = (M_1, M_2) \in (\mathbb{G}_1^*)^2$ of equivalence class $[M]$, and a secret key $sk = (x_1, x_2)$, choose $y \xleftarrow{\$} \mathbb{Z}_p^*$ and output $\sigma \leftarrow (Z, Y, \hat{Y})$ with

$$Z \leftarrow \left(\prod_{i \in [2]} M_i^{x_i}\right)^y \qquad Y \leftarrow P^{\frac{1}{y}} \qquad \hat{Y} \leftarrow \hat{P}^{\frac{1}{y}}. \tag{1}$$

$\texttt{Verify}(M, \sigma, pk)$: On input a representative $M = (M_1, M_2) \in (\mathbb{G}_1^*)^2$ of equivalence class $[M]$, a signature $\sigma = (Z, Y, \hat{Y}) \in \mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^*$, and a public key $pk = (\hat{X}_i)_{i \in [2]} \in (\mathbb{G}_2^*)^2$, check whether

$$\prod_{i \in [2]} e(M_i, \hat{X}_i) = e(Z, \hat{Y}) \qquad \wedge \qquad e(Y, \hat{P}) = e(P, \hat{Y}). \tag{2}$$

If this holds, output 1 and 0 otherwise.

$\texttt{ChgRep}(M, \sigma, f, pk)$**:** On input a representative $M = (M_1, M_2) \in (\mathbb{G}_1^*)^2$ of equivalence class $[M]$, a signature $\sigma = (Z, Y, \hat{Y}) \in \mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^*$, the randomness $f \in \mathbb{Z}_p^*$ and a public key pk, return $\perp$ if $\texttt{Verify}(M, \sigma, pk) = 0$. Otherwise pick $\psi \xleftarrow{\$} \mathbb{Z}_p^*$ and return (M^f, σ') with $\sigma' \leftarrow (Z^{\psi f}, Y^{\frac{1}{\psi}}, \hat{Y}^{\frac{1}{\psi}})$.

4.2 Provable Reward Computation

Now that we have covered SPS-EQ signatures (and, in particular, their verification function), we can proceed to the provable computation of the reward. At this point we will assume that the user has already interacted with the ads, and that the vector is already generated. Moreover, as discussed in the RFC&C report, the policy vector, $\boldsymbol{p} \in \mathbb{Z}_p^N$, is publicly available. Let the token and signature owned by the user be represented by:

$$\tau = (\tau_1, \tau_2) = (C^R, P^R) = \left(\left(h_1^{id} \cdot h_2^r \cdot \tilde{h}_1^{c_1} \cdots \tilde{h}_N^{c_N}\right)^R, P^R\right) \qquad \sigma = (Z, Y, \hat{Y})$$

with $\boldsymbol{c} \in \mathbb{Z}_p^N$, and R the randomisation stored throughout the protocol. At the time of redemption, the user first de-ransomizes the token, and adapts the signature to the new representation, mainly, it computes $\texttt{ChgRep}(\tau, \sigma, R, pk)$.

Next, to generate the request, the user needs to disclose the identifier of the token, compute the inner product between the counter vector and the policy vector, prove that the addition of all counters does not exceed a limit set by Brave, L, and generate a ZKP of correctness. Let $\texttt{Res} = \langle \boldsymbol{c}, \boldsymbol{p} \rangle$. The user generates the following proof:

$$\Pi = SPK\{(r, c_1, \ldots, c_N) :$$
$$C = h_1^{id} \cdot h_2^r \cdot \tilde{h}_1^{c_1} \cdots \tilde{h}_N^{c_N} \wedge \sum_{i=1}^{N} c_i < L \wedge \texttt{Res} = \langle \boldsymbol{c}, \boldsymbol{p} \rangle \wedge \tau = (C, P)\}$$

Note that it is safe to link the reward request to the user, as the only information leaked is the actual reward earned (this is an information we deem to be safe to leak at this stage). The common input of the proof consists of the token identifier, the token, the limit of interactions and the policy vector.

Reducing the complexity of the ZKP

The range proof that we include in the ZKP is there to rate-limit the number of rewards a user can claim. A weaker, but cheaper, restriction could replace this range proof, mainly checking that the result is smaller than the biggest payout times L. This modification makes Brave lose a bit of granularity on how to rate-limit requests, but it also considerably reduces the ZKP complexity. It would be interesting to understand the loses, in terms of computation and communication complexity, of enforcing the range proof *within* the ZKP.

4.3 Verifier's Perspective, and the Scalability Challenge

Now that we have seen what the reward request looks like, we can go forward with the description of the verifier's perspective. This is where the scalability challenge comes into play. The verification procedure of the reward calculation proceeds in two phases:

ZKP verification The verifier begins by checking the zero knowledge proof

$$\Pi.\texttt{Verif} \overset{?}{=} \top$$

SPS-EQ verification Next, the verifier checks if the token used in the ZKP has a valid signature

$$\texttt{Verify}(\tau, \sigma, pk) \overset{?}{=} 1$$

Note that the user, at the time of reward request, opens the identifier of the token. This will be sufficient to mark the token as used, and link it to the corresponding user to make the payment.

The scalability challenge consists in performing this verification procedure in a decentralized and scalable way. Depending on the project, the scalability solutions will differ, and we can only give some intuition of how this could be achieved. Some teams might directly implement the verification procedure on-chain, while others might use proof recursion, to batch into a single proof several reward requests verifications; this means verifying several SPS-EQ signatures in the recursion proof.

We are working on an implementation of the SPS-EQ scheme (available on github), which can be used as part of the scalability challenge. The library is a work-in-progress, so comments on the design of the API or parameter choice are welcome. Given the complexity of verifying SPS-EQ signatures (evaluating several pairings), in the next section we present a list of some possible alternatives for this BBA construction.

Removing pairings from on-chain execution

The verification procedure of the SPS-EQ signature might induce a considerable overhead for on-chain computation due to the number of pairings required in that operation. However, at the time of redemption, SPS-EQ does not give us any indispensable properties — we only require them for confirmation events. Moreover, we already assume that Brave is the issuer of the BBA. Therefore, one possible way of reducing the verification procedure is to have Brave centrally verify the SPS-EQ and sign the token using a different signature scheme (e.g. Schnorr). Now the user can make the reward request on-chain without an expensive signature verification procedure.

5 Alternative Verification Procedure

The construction of BEKSS makes use of a nice trick to make the size of the token, and show event, minimal with respect to prover's computation and communication complexity. However, it induces a considerable overhead to the verifier, as the latter needs to evaluate several pairing functions — this may become even more of a challenge when verifying signatures in a decentralized manner. To this end, we list other proposals that could be used, and the main reasons for why we believe they are not ideal.

Original BBA construction [1] This construction relies on Groth-Sahai proofs for the show event. It has several downsides, as it imposes a considerable overhead on the user, both in computation and communication complexity. Moreover, the verification procedure also requires the evaluation of pairings.

Updatable Anonymous certificates [8] The particular instantiation presented in this paper (Appendix E) based on Pointcheval Sanders Blind Signatures is quite interesting. However, the major downside is that the user needs to provide a zero knowledge proof of ownership of a valid signature at each show event. This proof is the main concern, as it needs to be computed, and sent, very often. If one manages to reduce the communication and computation complexity of generating such a proof, then this solution would also be an ideal candidate.

BBAs are not a technical requirement *per-se*. They are just the best tool we have found to build the design we are seeking — only the issuer can update the interaction vector, but do so in a privacy preserving manner. If you believe there are better-suited tools, feel free to contact us directly to discuss them further.

Acknowledgements

We thank Jan Bobolz and Sam Kim for going through the document and their useful comments.

References

[1] Tibor Jager and Andy Rupp. Black-box accumulation: Collecting incentives in a privacy-preserving way. *Proc. Priv. Enhancing Technol.*, 2016(3):62–82, 2016.

[2] Jan Bobolz, Fabian Eidens, Stephan Krenn, Daniel Slamanig, and Christoph Striecks. Privacy-preserving incentive systems with highly efficient point-collection. In Hung-Min Sun, Shiuh-Pyng Shieh, Guofei Gu, and Giuseppe Ateniese, editors, *ASIA CCS '20: The 15th ACM Asia Conference on Computer and Communications Security, Taipei, Taiwan, October 5-9, 2020*, pages 319–333. ACM, 2020.

[3] Jan Camenisch and Markus Stadler. Efficient Group Signature Schemes for Large Groups (Extended Abstract). In *CRYPTO*, 1997.

[4] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic macs and keyed-verification anonymous credentials. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1205–1216. ACM, 2014.

[5] David Pointcheval and Olivier Sanders. Short randomizable signatures. In *Proceedings of the RSA Conference on Topics in Cryptology - CT-RSA 2016 - Volume 9610*, page 111–126, Berlin, Heidelberg, 2016. Springer-Verlag.

[6] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2004.

[7] Christian Hanser and Daniel Slamanig. Structure-preserving signatures on equivalence classes and their application to anonymous credentials. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 491–511. Springer, 2014.

[8] Johannes Blömer, Jan Bobolz, Denis Diemert, and Fabian Eidens. Updatable anonymous credentials and applications to incentive systems. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 1671–1685. ACM, 2019.