Why Webstrates doesn't have fancy crypto

Alexandra Hou - 201400573
Cryptographic Computing Projects
Aarhus University

1 Abstract

This paper will look at the possibility of using a functional signature scheme as the underlying cryptographic primitive in an access right model for the research project Webstrates.

On a theoretical level, the functional signature scheme provides the abilities aspired in the suggested access right model, but given the implementation of Webstrates, along with the current implementations of zk-SNARKS the model presented in this paper is not yet feasible.

2 Introduction

Webstrates is a reseach project, that embodies a different way of thinking about software than what we are used to. Thinking about software differently also implies a change in how we implement security.

The short introduction to Webstrates taken from [Web] states: *Webstrates is a research project and an experimental system that we have designed to explore what we call shareable dynamic media: a software vision where the distinction between application and document is blurred and that treats collaboration, sharing, and distribution across heterogeneous devices as the norm rather than an exception. Shareable dynamic media are collections of information substrates (or substrates for short). Substrates embody content, computation, and interaction, effectively blurring the distinction between document and application. Substrates can evolve over time and shift roles, acting as what are traditionally considered documents in one context and applications in another, or a mix of the two. Webstrates (web substrates) allow us to explore this vision through a simple yet powerful change to basic web technology.*

Since webstrates is still so new, there has not yet been put a time and effort in the security aspects of the system. As mentioned in [Klo+] *A more refined authentication and access rights model is left for future work.* This is the main focus of this paper, to try creating an access right model, deciding what cryptographic primitives can be used, and the feasablity of implementing this model.

3 Identifying security issues

In order to identify what sort of cryptography that could be relevant in Webstrates we started out by looking at the security issues that are pressing at the moment. This was done in collaboration with Clemens Klokmose, James Eagan, Kristian Antonsen, and Claudio Orlandi. The findings of this meating, was that the main problem in Webstrates in its current form is concerning the lack of an access right model. At the moment Webstrates support read access and read/write access to a webstrate. This means given read/write access to a webtstrate you can do anything. Obviously we do not want everyone to be able to do anything, since this opens up a variety of nasty security issues. Instead we would like differentiating rights to different part of a webstrate, e.g. if we have a comment section in a webstrate, everyone should be able to write text, and only text, in this section, but not nessesarily anywhere else in the webstrate.

Since webstrates are organized as tree structures we can place the rights for subtrees of a node in the node, making it possible to calculate the validity of an operation given the placement of the operation in the tree.

Since all clients will have access to the tree of a webstrate the server and all the clients are equally able to calculate the validity of an operation.

3.1 An initial sketch of an access right model

The primary function of the server are ensuring eventual consistency for all clients of a webstrate. Calculating the validity of an operation would slow the server down unreasonably, and falls outside the main responsibility of the server. The responsiveness of the server should not exceed 1 ms.

The main idea for accomodating the issue, is to make the client calculate the validity of the operation, and send some tag to the server along with the operation, and let the server verify this calculation using that tag.

In order for this to make sense, verifying the calculation using the tag should of course be much faster than actually doing the calculation itself, otherwise we would only contribute to the work of the server, instead of trying to minimize it.

4 Functional signature scheme as a solution

A succinct function private signature scheme as described in [BGI13] has the qualities laid out in the initial sketch of a solution.

The basic idea of the functional signature scheme is that only messages that exists in the range of some function f can be signed, and thereby it is possible to predefine what messages are valid. We can e.g. design this function to be

$$f(m) = \begin{cases} m \text{ if } P(m) = 1 \\ \perp \text{ otherwise} \end{cases}$$

Where P is some predicate that outputs 1 on messages that satisfies the given policy.

This section will briefly go through the steps used to create a succinct function private functional signature scheme using SNARKs based on the description in [BGI13].

4.1 SNARKs

A SNARK is a succinct non-interactive argument of knowledge and can be described by the tuple $\Pi = (Gen, Prove, Verify, S, E)$ where Gen produces some private information, and some public information (a crs). $Prove$ generates a proof π that $x \in L$ given the public information, a word x, and a witness w for $x \in L$. $Verify$ checks given the secret information, the word x, and the proof π that π is a proof that $x \in L$. E being the extractor, and S the algotithms used to ensure adaptive zero-knowledge of the SNARK. [BGI13]

4.2 Standard signature scheme

A signature scheme over a message space M can be described as the tuple $S = (S.Gen, S.Sign, S.Verify)$ where

$S.Gen(1^k) \rightarrow (sk, vk)$

$S.Sign(sk, m) \rightarrow \sigma$, σ being the signature on message m

$S.Verify(vk, m, \sigma) \rightarrow \{0, 1\}$, the verifying algorithm will return 1 when σ is a correct signature for message m.

This scheme upholds correctness and unforgability under chosen message attack, for a formal proof of this see [BGI13].

4.3 Functional signature scheme from one way functions

By the use of the standard signature scheme and the assumption that one way functions exist we can thereby get a functional signature scheme described by the tuple $FS0 = (FS0.Setup, FS0.Keygen, FS0.Sign, FS0.Verify)$ where

$FS0.Setup(1^k)$
$S.Gen(1^k) \rightarrow (msk, mvk)$
output (msk, mvk)

$FS0.Keygen(msk, f)$
$S.Gen(1^k) \rightarrow (sk, vk)$
$S.Sign(msk, f|vk) \rightarrow \sigma_{vk}$ $c = (f, vk, \sigma_{vk})$
output $sk_f = (sk, c)$

$FS0.Sign(f, sk_f, m)$
$S.Sign(sk, m) \to \sigma_m$
$\sigma = (m, c, \sigma_m)$
output $(f(m), \sigma)$

$FS0.Verify(mvk, m*, \sigma)$
output 1 if $m* = f(m)$
$S.Verify(vk, m, \sigma_m) \to 1$
$S.Verify(mvk, vk|f, \sigma_{vk}) \to 1$

The unforgability of this scheme is based on the unforgability of the underlying signature scheme, for a formal proof see [BGI13]. Clearly this is not function private nor succinct, and verification requires the calculation of the function f on message m, which renders it useless in most applications.

4.4 Succinct function private signature scheme from SNARKs

Given an unforgable functional signature scheme FS0, and a an adaptive zero-knowledge SNARK Π for the language
$L = \{(m, mvk)|\exists\sigma \text{ s.t. } FS0.Verify(mvk, m, \sigma) \to 1\}$
We can construct a succinct function private signature scheme as seen in [BGI13] where

$FS.Setup(1^k)$
$FS0.Gen(1^k) \to (msk, mvk')$
$\Pi.Gen(1^k) \to crs$
$mvk = (mvk', crs)$
output (msk, mvk)

$FS.Keygen(msk, f)$
$FS0.Keygen(msk, f) \to sk_f$
output sk_f

$FS.Sign(f, sk_f, m)$
$FS0.Sign(f, sk_f, m) \to \sigma'$
$\pi = \Pi.Prove((f(m), mvk'), \sigma', crs)(m, c, \sigma_m)$
output $(m* = f(m), \sigma = \pi)$

$FS.Verify(mvk, m*, \sigma)$
output $\Pi.Verify(crs, m*, \sigma)$

The zkSNARK is used so that when signing a message, there's generated a proof that $(f(m), mvk') \in L$, and verification is then reduced to verifying that σ is a valid argument of knowledge of a signature of $f(m)$ in the underlying functional signature scheme.

The unforgability of this scheme is again reliant on the unforgability of the underlying functional signature scheme, function privacy is given since it can be shown that an adversary who succeeds in a function privacy game with noticable advantage can be used to break the zk property of the SNARK. Succinctness followes directly from the succinctness property of the SNARK, for further elaboration of this, and formal proofs see [BGI13].

4.5 Usability in Webstrates

The first intuitive sketch of a solution was to make the client calculate the validity of the operation, and send some tag to the server along with the operation, and let the server verify this calculation using that tag.

The succinct function private signature scheme described above does exactly this. By using a predicate P as described in the beginning of this section $f(m) = m$ when m is a valid message, which means that the information sent to the server will be the message m, and the proof/signature σ, where the proof that is the output of the signing algorithm will act as the tag.

If we define a function, satisfying the rules that we want for each node in the webstrate, the client will then be able to sign an operation sent to the server using the function stored in the parent node of the operation made. The server now only needs to run the verification algorithm to assess weather or not this is a valid operation.

The succintness of the scheme will ensure that verifying the validity of an operation, is independant of the size of f and the size of m, but in this case where f is not that large, this might be slower that actually calculating f(m). But in this case the verificaton process could be used to not only ensure validity of operations, but also ensure validity of the code written.

For this to work, it will of course require there to be some setup and key-generation when logging in to a webstrate for the first time.

5 Feasability of this solution

Since the presented functional signature scheme uses the Verification algorithm for the SNARK as the verification algorithm for the functional signature scheme, we can look at the verification time for a SNARK as a lower bound for the verification time of the functional signature scheme. Also proof times are very high and depending on how exactly the proofs will be generated, and how much can be done in an offline setup phase and reused, these might make it unfeasable.

If we look at Pinoochio[BP13], a deployed system for SNARKs, we get a verification time of approximately 10 ms. Looking at other implementations such as libsnark[Lib], we can get verification times as low as 5 ms.[BS+13]

Given the constraints on server response time of 1 ms, a verification time of 5-10 ms will not live up to these constraints.

6 Conclusion

Given the current implementations of webstrates and the advancement in creating SNARKs we are still not in a place where the running time of verifying a SNARK is fast enouogh for use in webstrates. Since this is a very fast developing area in cryptography, especially because of it's use in verifiable delegation of computation, it might improve substantially within a reasonable time frame, making this access right model usable, but until then another model will have to be set up in order to handle access rights in webstrates.

7 Future work

The use of SNARKs in the functional signature scheme makes it possible to create functional signatures from any predicate, on the message m to be signed, but in this case we don't need it to work for every predicate, but only for some. It would be interesting to look into the possibility of a functional signature scheme that only works for a subset of predicates.

8 Acknowledgements

Thanks to Clemens Klokmose, James Eagan, Kristian Antonsen, and Claudio Orlandi who played a crucial part in defining the subject of this paper. An extra thanks to Claudio Orlandi who has been the supervisor on this project.

References

[BGI13] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. *Functional Signatures and Pseudorandom Functions*. Cryptology ePrint Archive, Report 2013/401. `http://eprint.iacr.org/2013/401`. 2013.

[BP13] Craig Gentry Mariana Raykova Bryan Parno Jon Howell. "Pinocchio: Nearly Practical Verifiable Computation". In: *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 2013. URL: `https://www.microsoft.com/en-us/research/publication/pinocchio-nearly-practical-verifiable-computation/`.

[BS+13] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. *Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture*. Cryptology ePrint Archive, Report 2013/879. `http://eprint.iacr.org/2013/879`. 2013.

[Klo+] Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. *Webstrates: Shareable Dynamic Media*. `http://www.klokmose.net/clemens/wp-content/uploads/2015/08/webstrates.pdf`.

[Lib] *libsnark*. `https://github.com/scipr-lab/libsnark`. 2016.

[Web] *webstrates.net*. `http://www.webstrates.net`. 2016.