A bit of an update from ‘Dark Crystal’. We are continuing to develop our identity recovery system, working on ‘forwarding’ shares to people other than the author of the secret. Parallel to this we are currently researching two areas. Adopting secp256k1 keys, and security issues/vulnerabilities with Shamir’s secret sharing and how to approach them. This article is about the latter, and is heavily inspired by a recent security review from Dominic Tarr. We hope that this will be useful for other projects considering using Shamir’s scheme for identity recovery.

Note:

As a new use I am apparently only allow to add two links to posts, so i had to cut the links out of the references.

Security considerations for Shamir’s secret sharing

In general, Shamir’s scheme is considered information-theoretically secure. That is, individual shares contain absolutely no semantic information about the secret, and it can be said to be ‘post quantum’ cryptography.

An interesting anecdote, the root key for ICANN DNS security, effectively the key which secures the naming system of the internet, is held by seven parties, based in Britain, the U.S., Burkina Faso, Trinidad and Tobago, Canada, China, and the Czech Republic. Cryptographer Bruce Schneier has alleged that they are holders of Shamir’s secret shares, which indicates the scheme is taken quite seriously.

However, it is not without its problems:

The need for verification of individual shares

Harn and Lin consider the situation in which ‘cheaters’ claiming to be holders of shares introduce ‘fake’ shares, causing the incorrect secret to be recovered. Of course without having the other shares they have no control over the content of the ‘incorrect’ secret.

This is not so much of a concern for us as we already have a way to validate who is a custodian. However we also considered the possibility that genuine holders of shares might have a motivation for not wanting the secret to be recovered, and could maliciously modify their share. Furthermore, the shares might be modified by some accidental or external cause, and it is important to be able to determine which share is causing the problem.

It might be very easy to determine that we have recovered the wrong secret. Either because we have some idea of how we expect it to look, or as we have recently implemented in dark crystal, an identifier is added to the secret to allow the correct secret to be automatically recognised. (We concatonate the secret with the last 16 bytes of its SHA256 hash).

However, the problem here is that although we might know for sure that we have not successfully restored our secret, we have no way of telling which share(s) have caused the problem, meaning we do not know who is responsible.

The solution is to introduce some verification of shares, and a number of different methods of doing this have been proposed. Typically, they rely on publicly publishing some information which allows verification of a given share.

Here are some possible solutions:

Publicly publishing the encrypted shares

This is what we were originally planning to do, but this only helps in this context if the encryption scheme used is deterministic. That is to say encrypting the same message with the same key twice will reliably give the same output. The problem here is that such encryption schemes are vulnerable to replay attacks. Most modern asymmetric schemes introduce some random nonce to evade this problem. The scheme we are using (libsodium’s secret box) typically takes a 24 byte random nonce. So this is not a good option. However we actually already ‘wrap’ shares in a second level of encryption to to allow the secret owner to publish their own encrypted copy of the share message with its associated metadata as a way to keep a personal record of what was sent to who. When we looked at other schemes for verifiable secret sharing we found they involved a similar practice, and we decided to use an existing well-documented scheme.

Publicly publishing the hash of each share

This is also something we considered, but feel that it gives custodians more unnecessary extra information and less accountability compared to other methods.

Feldman’s scheme

Paul Feldman proposed a scheme in 1987 which allows custodians to verify their own shares, using homomorphic encryption (an encryption scheme where computation can be done on encrypted data which when decrypted gives the same result as doing that computation on the original data) on top of Shamir’s original scheme.

Schoenmakers scheme

More recently Berry Schoenmaker proposed a scheme which is publicly verifiable (originally introduced by Stadler, 1996). That is, not only custodians, but anybody is able to verify that the correct shares were given. The scheme is described in the context of an electronic voting application and focusses on validating the behaviour of the 'dealer' (the author of the secret). But it can just as well be used to verify that returned shares have not been modified, which is what we are most interested in.

Implementations

We are currently considering the following implementations:

- github songgeng87/PubliclyVerifiableSecretSharing - C implementation built on secp256k1 used by EOS
- github FabioTacke/PubliclyVerifiableSecretSharing - A Swift implementation
- github dfinity/vss - Dfinity's NodeJS implementation built on BLS, and used for their distributed key generation

However, these do not give a drop-in replacement for the secrets library we currently use. Adopting verifiable secret sharing would require a large change to our codebase and mean we need to reconsider several aspects of our model. But it would bring a great advantage in terms of security.

Share size has a linear relationship with secret size

Anyone holding a share is able to determine the length of the secret. Particular kind of cryptographic keys have a characteristic length. So the scheme gives away more information to custodians than is necessary. Our solution to this is to add padding to the secret to increase share length to a standard amount.

Revoking shares if trust in a custodian is lost

Suppose we loose trust in one person holding a share. This might be because they had their computer stolen. Or maybe we had a really bad argument with them. Or maybe we found out they weren't the person they were claiming to be.

In Shamir's original paper he states that one of the great advantages of the scheme is that it is possible to create as many distinct sets of shares as you like without needing to modify the secret. Each set of shares is incompatible with the other sets. Using Glenn Rempe's implementation, if we run the share method several times with the same secret, we get each time a different set of shares. When generating a new set, an extra check could be done to rule out the extremely improbable case that an identical set had been generated.

This means in a conventional secret sharing scenario (imagine the shares are written on paper and given to the custodians), we could simply give new shares to the custodians we do still trust and ask them to destroy the old ones. This would make the share belonging to the untrusted person become useless.

In our case, we are using Secure-Scuttlebutt's immutable log, and have no way of destroying a message. A solution we are considering, is to use ephemeral keys which are used only one time for a particular share and can be deleted when a new set of shares is issued. This gives greatly increased security, but the cost is more keys to manage and increased complexity of the model.

Secure computation

Having a good system of encryption does not give us security if the host system is compromised. We are considering using a dedicated virtual machine for secure computation, such as Dyne.org's 'Zenroom'. However there, are many more considerations one needs to make, especially if secret is initially stored on disk. But this goes beyond the scope of assessing the security of Shamir's scheme.

Conclusion

We feel confident that we are able to address the issues we have explored in this article, although not all of them will be implemented in the next release. However, we have focussed here mainly on technical limitations of the scheme. There are many other social aspects which pose threats to our model, which we will explore in another article.

References

- Beimel, Amos (2011). "Secret-Sharing Schemes: A Survey" cs.bgu.ac.il/~beimel/Papers/Survey.pdf
- Blakley, G.R. (1979). "Safeguarding Cryptographic Keys". Managing Requirements Knowledge, International Workshop on (AFIPS). 48: 313–317. doi:10.1109-/AFIPS.1979.98.
- Feldman, Paul (1987) "A practical scheme for non-interactive Verifiable Secret Sharing" Proceedings of the 28th Annual Symposium on Foundations of Computer Science

- Harn, L. & Lin, C. Detection and identification of cheaters in (t, n) secret sharing scheme, Des. Codes Cryptogr. (2009) 52: 15.
- Schneier, Bruce (2010) - DNSSEC Root Key held by 7 parties worldwide
- Schoenmakers, Berry (1999) “A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting” Advances in Cryptology-CRYPTO’99, volume 1666 of Lecture Notes in Computer Science, pages 148-164, Berlin, 1999. Springer-Verlag.
- Shamir, Adi (1979). “How to share a secret”. Communications of the ACM. 22 (11): 612–613. doi:10.1145/359168.359176.
- Zenroom, a virtual machine for fast cryptographic operations on elliptic curves

See Also…

- Our ‘list of applications and articles on Shamir’s secret sharing’
- ‘Brainstorming Coconut-related scenarios’ (‘Coconut death’ refers to a role playing game we did as part of our research where we tried to recover the keys of members of the group who had been hit by coconuts)
- ‘Thoughts on verifying received shards in dark crystal’
- You can also follow our development on Secure Scuttlebutt channels #darkcrystal

#dark-crystal

and #mmt