Scope

The target of the audit was two golang projects:

- git revision ce779395f4c98898f21f8c49f71f4b3353995127 of https://github.com/privacybydesign/gabi/
- git revision 36ab84c97ac789aa4df4a03fbf7ea66fe6ab341e of https://github.com/privacybydesign/irmago/

gabi implements the cryptographic core of IRMA - a slight variant of the IDEMIX protocol. We verified that the implementation actually matches the specification as given. During our gabi audit we primarily focused on information leakage.

irmago implements a higher layer on top of gabi enabling the protocol to be used by UI frontends. Our focus was logic bugs and any bugs that violate availability and the security properties of idemix.

Team

The audit was conducted by:

- Jonathan Levin
- Stefan Marsiske

Timespan

The audit was conducted between 2020-02-01 and 2020-03-31

Results Summary

We found only six issues in total in gabi, the most critical ones being imposed by the language itself, which considers information leakage out of scope of memory safety and timing side channels out of scope of their cryptographic modules. This resulted in non-constant time exponentiations using secret key material. We also found some unnecessarily lax file system permissions on public and private keys stored by gabi. All reported gabi issues can be seen here: https://github.com/privacybydesign/gabi/issues?q=is%3Aissue+author%3Astef

In irmago we found a few interesting issues revolving around the updating of scheme managers. Timestamps on schemes were not signed (and therefore also not checked for validity), providing a downgrade attack to old schemes. Furthermore, in the presence of a successful MiTM, a TOCTOU issue during scheme updates can also lead to an arbitrary file write over the network attack.

We also found the authentication around the keyshare server to be comparatively simple, relying on a single SHA-256 hash of the user's PIN. We found a simple

way to DoS users out of their accounts by maxing out the PIN attempts of another user. We encountered some low importance issues like the recurring theme of unbounded reading from file descriptors. Finally, we suggested improvements to the TLS configuration specified in irmago, which did not support all TLSv1.3 ciphersuites.

Methodology

We did a thorough reading of all the source code. First we conducted a breadth-first approach, highlighting interesting/security-sensitive locations in the code, and noting questions we had. We consulted regularly with the Project Manager of the IRMA project Sietse Ringers to verify our understanding. After the first pass we went into each noted location and examined them in depth, writing test code to verify expected behavior when needed. We also used some static analysis tools such as gosec (https://github.com/securego/gosec) and staticcheck (https://staticcheck.io/). These static analysis tools did not reveal any novel issues.

Conclusion

The choice to use golang eliminated the bug class of buffer overflows. While this is definitely a good thing, it also introduced issues regarding leakage of sensitive information (using https://github.com/awnumar/memguard for protecting sensitive data in RAM is warmly recommended). This also provides the RU with a low-hanging fruit of implementing time-safe modular exponentiation, multiplication and GCD for golang or use and collaborate with https://github.com/coyim/constbn/ establishing a robust base for cryptographic math with constant time requirements. With respect to the higher level framework provided by irmago, there are a few functions that use keys and other cryptographic functionalities that are not part of the Idemix protocol, and these lack a proper key-lifecycle or modern primitives (e.g., the simple pin-based authentication scheme towards keyshare servers).

Appendix

Our notes that we took during our audit can be found in the accompanying org-mode document which can be found here in HTML format.