Java Cryptographic Engine (JCE) and Bouncy Castle Crypto API to implement privacy CA. The privacy CA application is running on the server. To measure the performance, we first tested the login without remote attestation, simply using user name and password. The average time for authentication was 0.015 seconds. We then use login with remote attestation. For the purpose of remote attestation, we used RSA 2048 bit and AES 256 bit with CBC attributes for public and symmetric key encryption respectively. SHA-1 with 160-bit was used as the hash function. We obtained the random numbers from the RNG function of the TPM chip. We measured that, on average, using remote attestation took 7.628 seconds, compared to 0.015 seconds using plain user name and password combination. We believe that this overhead was due to the following:

- Device initialisation, required data collection and creation of the AIK cryptographic key: average 2.39 seconds;
- Certificate creation by the Privacy CA: average 1.96 seconds;
- Certificate activation after verifying the environment the key has created has not changed: average 3.27.

In short, the overhead is the cost of “hardening” the login process using TPM, where most of the time is spent on executing TPM functions and attestation. An analysis on attestation based authentication is also reported in [50].

Further analysis of the attestation protocol revealed that 39% time was spent on making TPM function calls via the TCG software stack (TSS), whereas 61% time was spent on non-TPM related function calls such as connection to remote hosts. We further analysed the time spent on TPM calls and identified four critical function calls in the mutual attestation protocol. This is done to ensure that there is no hidden overhead to any particular function calls to TPM chip other than the expense incurred to making TPM calls in general, as well as possibility of optimizing TPM calls. These are: CollateIdentity, IdentityCredential, Quote, and VerifyIdentityCredential. Out of time spent on TPM calls, CollateIdentity took 48% time, IdentityCredential took 19%, Quote took 19% and VerifyIdentityCredential took 14%. We conducted further analysis of implementation code with the aim of minimizing the number of TPM calls and the complexity of code at each stage. This led us to the conclusion that TPM calls are stable as each TPM call at different stages took about the same time without having particular overhead on any specific TPM calls because of different implementation code. For example, both IdentityCredential and Quote made about 12 TPM calls, whereas VerifyIdentity Credential made 9 TPM calls. CollateIdentity made 20 TPM calls, as well as complex calculation of bytes, to create and use credentials taking more time than calls used in other stages. These observations led us to believe that there is no overhead placed in any particular TPM calls in our implementation. However, if these calls are repeated for every message exchanged between USB-device and the enterprise server, the TPM calls become a bottleneck for efficient transfer of data between to and from USB device. From these simulation results, we observed that reducing a large number protocol messages that subsequently triggers a large number of TPM calls might result in an efficient attestation.