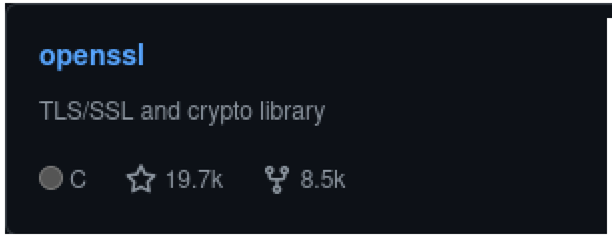
Alexandru Calotoiu

DaFlEx

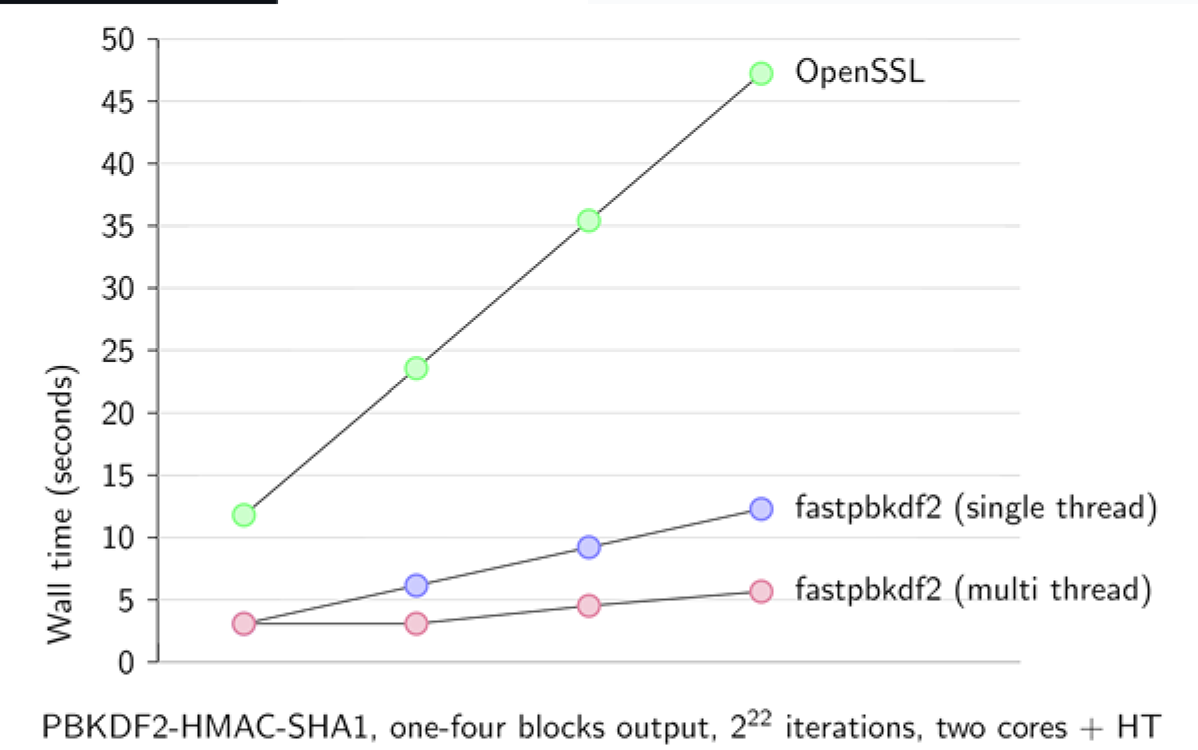
C2DaCe – an update



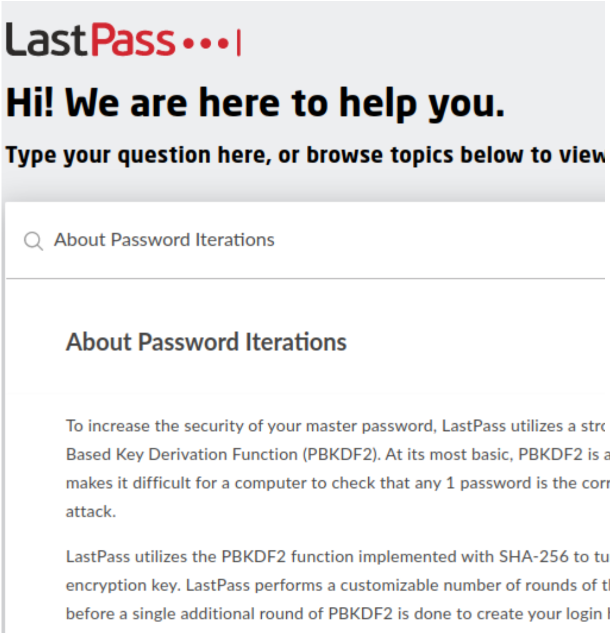
openssl

TLS/SSL and crypto library

● C ☆ 19.7k 8.5k



PBKDF2-HMAC-SHA1, one-four blocks output, 2^{22} iterations, two cores + HT



LastPass•••|

Hi! We are here to help you.

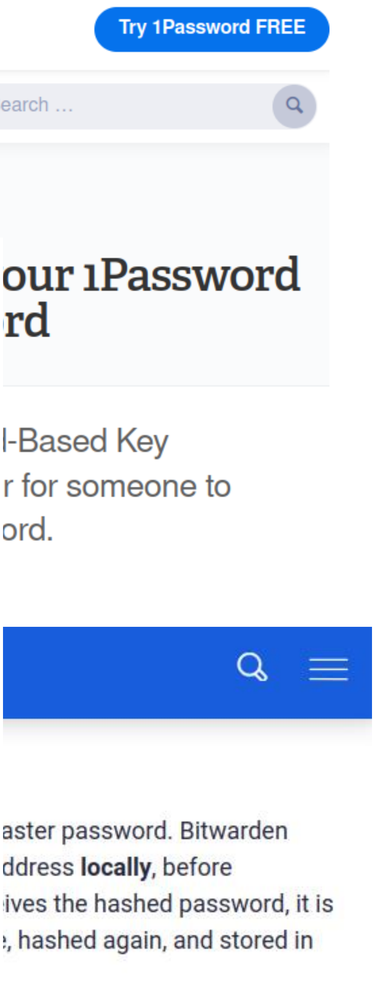
Type your question here, or browse topics below to view

About Password Iterations

About Password Iterations

To increase the security of your master password, LastPass utilizes a stro
Based Key Derivation Function (PBKDF2). At its most basic, PBKDF2 is a "password-strengthening algorithm" that makes it difficult for a computer to check that any 1 password is the correct master password during a compromising attack.

LastPass utilizes the PBKDF2 function implemented with SHA-256 to turn your master password into your encryption key. LastPass performs a customizable number of rounds of the function to create the encryption key, before a single additional round of PBKDF2 is done to create your login hash.



Menu

Try 1Password FREE

1Password Support

Contact Support

Search ...

SECURITY AND PRIVACY

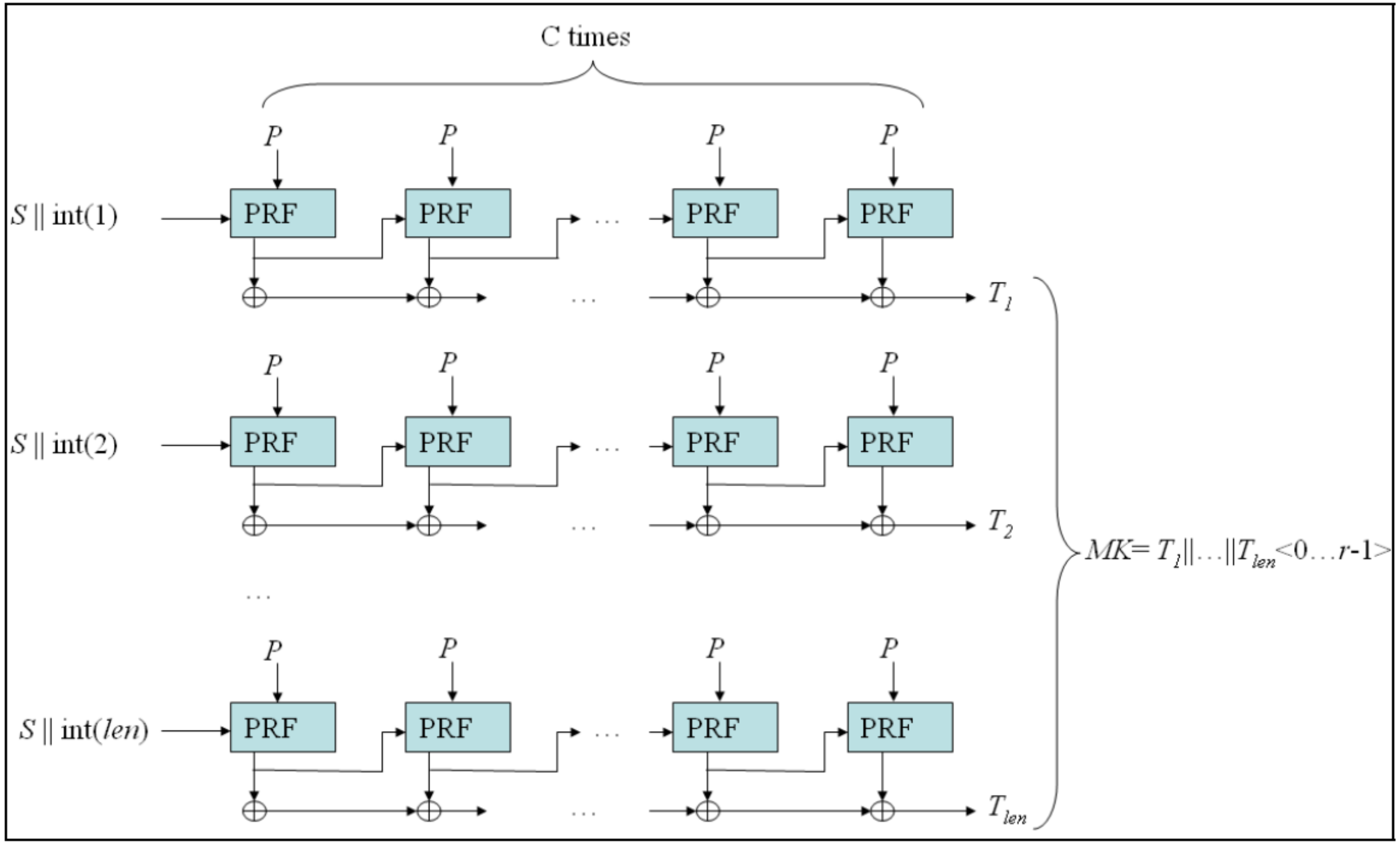
our 1Password
rd

l-Based Key
r for someone to
ord.

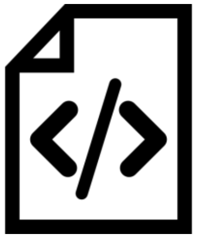
aster password. Bitwarden
ddress **locally**, before
ives the hashed password, it is
:, hashed again, and stored in

The default iteration count used with PBKDF2 is 100,001 iterations on the client (*client-side iteration count is configurable from your account settings*), and then an additional 100,000 iterations when stored on our servers (for a total of 200,001 iterations by default). The Organization key is shared via RSA-2048.

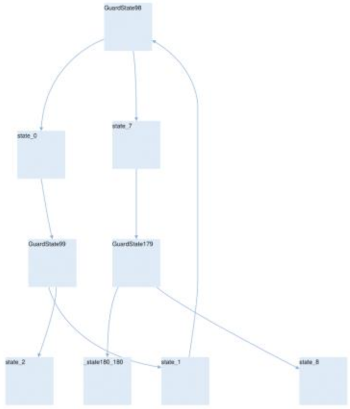
PBKDF2


C times
P
S || int(1)
PRF
. . .
T_1
S || int(2)
T_2
S || int(len)
T_{len}
$MK = T_1 || \ldots || T_{len}\langle 0 \ldots r\text{-}1 \rangle$

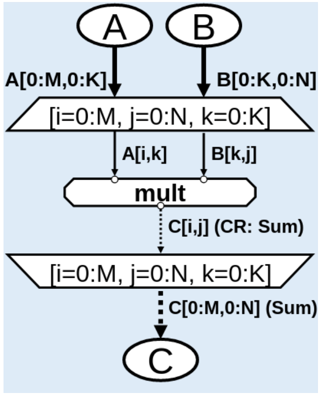
C2DaCe for PBKDF2



C code



Initial SDFG



Parallel SDFG

Parallel Executable

Status

- Extract the PBKDF2 implementation from OpenSSL and create a micro-benchmark (note that we will still use the SHA functions from OpenSSL as external calls).
- Add support to handle pointers to linear data by splitting the pointer into a data container and an offset variable.
- Handle the struct pointers used by OpenSSL (state pointers) to keep a state between the SHA API calls. This is done by creating data dependencies into the SDFG that follow the real dependencies needed to execute the SHA API calls successfully.
- Divide the state pointer dependencies into the requirement that the pointer was initialized and the real data dependency created by reading or writing to the state.
- Test and validate the correctness of the SDFG

- If needed, expand the LoopToMap transformation that identifies the parallelization opportunities. It usually acts on for loops but the loop inside PBKDF2 is a while loop, some tweaking might be needed.

- Test the performance of the resulting compiled SDFG