Implementation of Split Key STS-KDF Protocol*

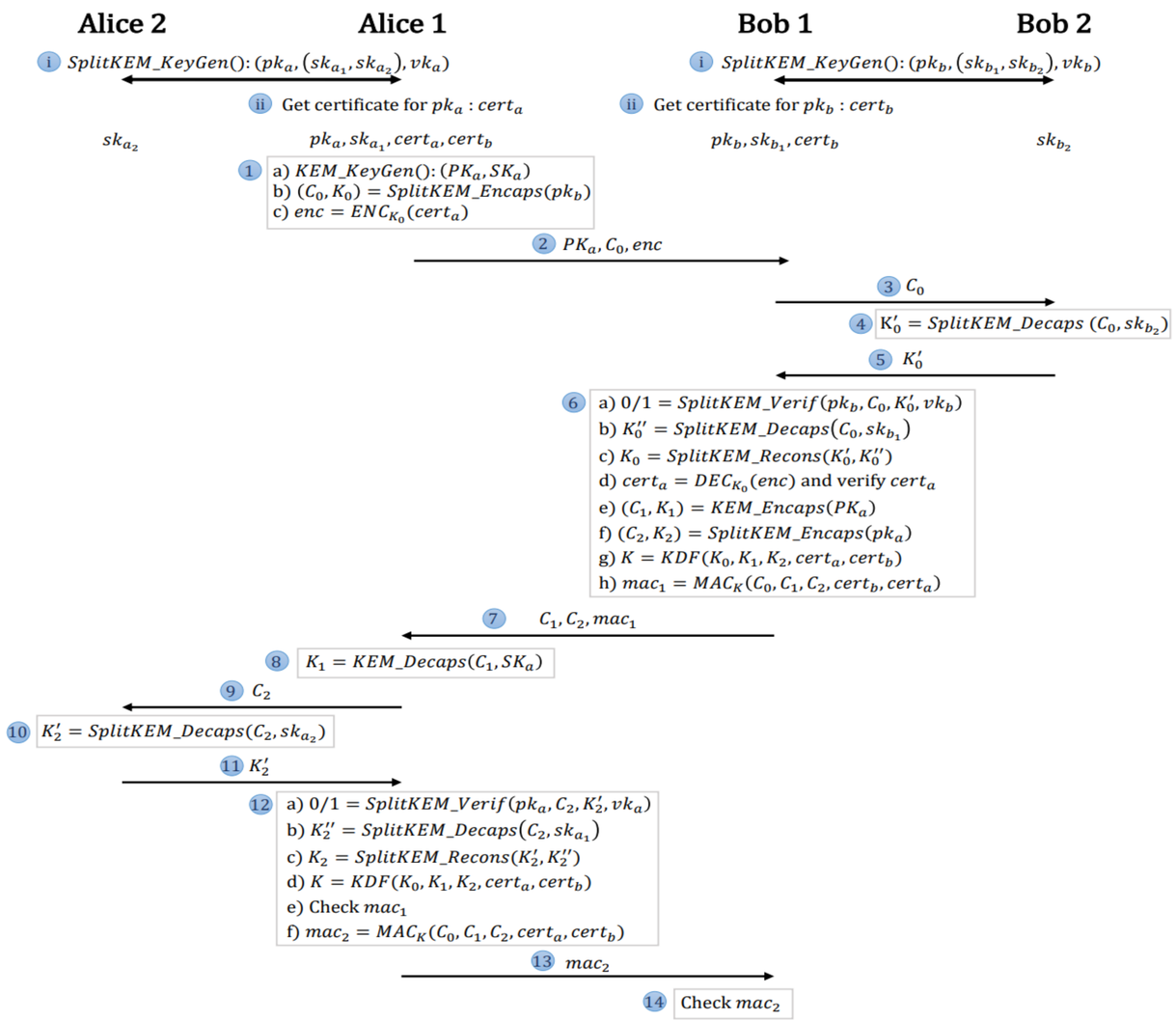
Joonas Ahola, Philip Ginzboorg, Sampo Sovio

*Akman et al., "Split keys for station-to-station (STS) protocols", Journal of surveillance security and safety, vol. 4, no.3, 2023.

Implementation Goals

- Working C implementation of a Split Key STS-KDF protocol.
- Good performance, to be used in performance-critical applications.
- Applicable to embedded devices and resource-constrained environments.
- Use **MbedTLS** crypto library.

- **MbedTLS** is a C library implementing SSL/TLS and DTLS protocols.
- Has a small code footprint and has options to reduce it even further for embedded development.




```c
#ifndef SPLIT_ECIES_H
#define SPLIT_ECIES_H

#include <stdio.h>
#include "mbedtls/bignum.h"
#include "mbedtls/ecp.h"
#include "mbedtls/md.h"
#include "mbedtls/cipher.h"

int point_x_bytes_secp256r1(mbedtls_ecp_group* mbedtls_group, mbedtls_ecp_point* sepc256r1_point, char* byte_buffer);
int point_y_bytes_secp256r1(mbedtls_ecp_group* mbedtls_group, mbedtls_ecp_point* sepc256r1_point, char* byte_buffer);

int bytes_to_point_secp256r1(mbedtls_ecp_group* mbedtls_group, mbedtls_ecp_point*  sepc256r1_point,
                             unsigned char* byte_buf_x, unsigned char* byte_buf_y);

int SplitKEM_KeyGen_secp256r1(mbedtls_ecp_group* mbedtls_group, mbedtls_mpi* mbedtls_key_A1,
                              mbedtls_mpi* mbedtls_key_A2, mbedtls_mpi* key_A,
                              int (*f_rng)(void*, unsigned char*, size_t), void *p_rng);

int SplitKEM_Reconst_secp256r1(mbedtls_ecp_group* mbedtls_group, mbedtls_ecp_point* ks_1,
                              mbedtls_ecp_point* ks_2, unsigned char* K, mbedtls_md_info_t* cipher_info);

int SplitKEM_Encaps_secp256r1(mbedtls_ecp_group* mbedtls_group, unsigned char* K,
                              mbedtls_ecp_point* pub_key_B, mbedtls_mpi* priv_key_A,
                              mbedtls_md_info_t* cipher_info, int (*f_rng)(void*, unsigned char*, size_t), void *p_rng);

int ECIES_enc(mbedtls_ecp_group* mbedtls_group, mbedtls_ecp_point* pub_key_B,
              unsigned char* plain_txt, size_t, unsigned char* cipher_txt,
              unsigned char* K, int (*f_rng)(void*, unsigned char*, size_t), void *p_rng);

int ECIES_dec(mbedtls_ecp_group* mbedtls_group, mbedtls_ecp_point* C,
              mbedtls_mpi* priv_key_B, unsigned char* cipher_txt, mbedtls_md_info_t* cipher_info,
              unsigned char* decrypted_txt, unsigned char* K, int (*f_rng)(void*, unsigned char*, size_t), void *p_rng);

#endif // SPLIT_ECIES_H
```

Future Work

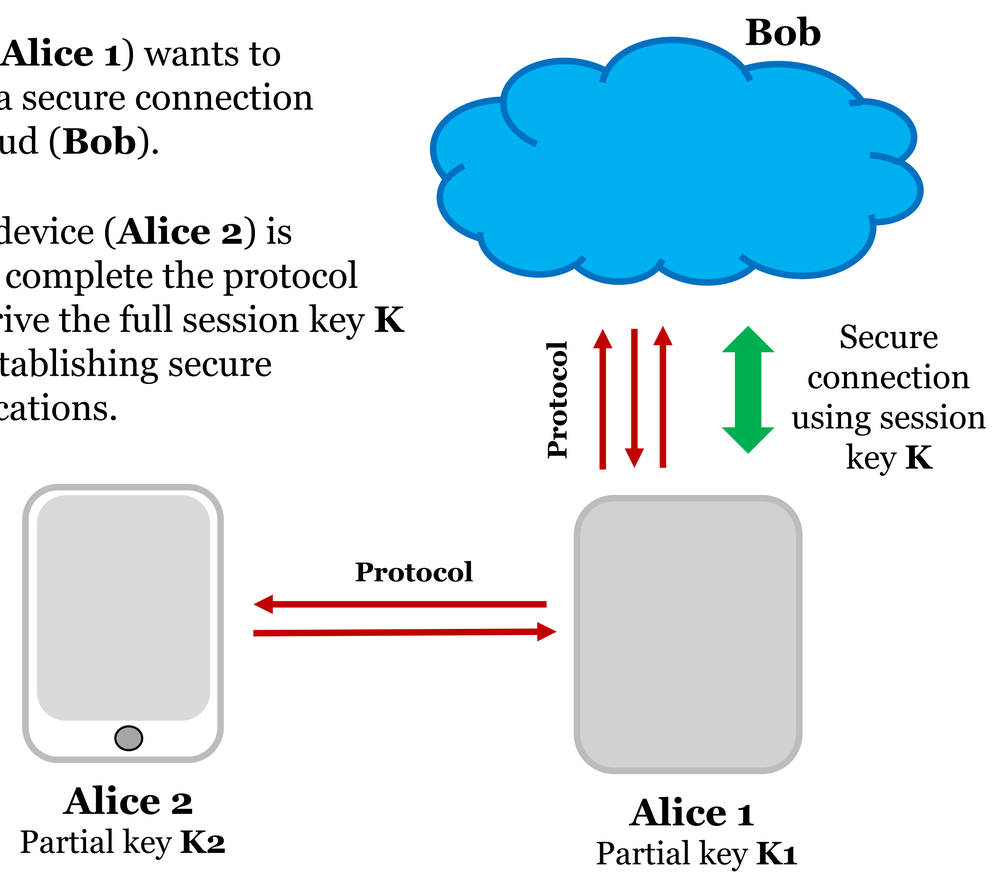
- Migrate to Post-Quantum protocols, for example CRYSTALS-Kyber for key encapsulation mechanism (KEM).

Split Key

- In situations where a secret key may be vulnerable to exposure, splitting the secret key between several devices so that those devices have to cooperate to use the key may reduce the risks of key exposure and provide an additional level of control over key usage.
- The key could be split between: IoT device-smartphone, wireless base-core network, or two different computers in a data center.

Possible Application of Split Key STS-KDF

- A device (**Alice 1**) wants to establish a secure connection with a cloud (**Bob**).
- A mobile device (**Alice 2**) is needed to complete the protocol and to derive the full session key **K** used in establishing secure communications.



STS-KDF Protocol with KEM*

- Station-to-Station: Authenticated key agreement (AK) protocol that combines the Diffie-Hellman (DH) key agreement and signature-based authentication.
- STS-KDF: A variant of STS protocol that have *explicit key confirmation* by using a symmetric-key encryption scheme and a message authentication code and prevents Unknown Key Share (UKS) attacks.
- Formal verification with ProVerif, proven Secrecy of **session key K**, Mutual Authentication on **K**, Forward Secrecy, Resistance to UKS, KCI and Reflection Attacks.

Comparisons with base implementation in Python and current implementation in C

Runtime in milliseconds, average of 50 runs

Function	C with MBedTLS	Python
ECIES_encryption	0.015 ms	31.5 ms
ECIES_decryption	0.04ms	64.2 ms
SplitKEM_Reconst.	0.0003 ms	0.22 ms

Lines of code

Function	C with MBedTLS	Python
ECIES_encryption	41	18
ECIES_decryption	47	8
SplitKEM_Reconst.	146	3

Helsinki System Security Lab (HSSL)

HSSL drives renewal and mastery in the field of platform and device related security technologies, especially for Huawei consumer devices such as mobile phones, laptops, televisions and automotive. We do research in topics such as hardware-assisted isolation and integrity, as well as in operating system protection (hypervisor, TEE, secure enclaves and kernel hardening). We also carry expertise in cryptography and systems security functionality such as device key management (PKI), device attestation and key-store solutions.

HUAWEI