Pseudokod pre protokol SAKE

Algorithm 1 SAKE: Inicializacia retazca klucov

function INITIALIZE_KEY_CHAIN(*master_key*, *is_initiator*)
    Input: *master_key* (32B tajomstvo), *is_initiator* (boolean)
    Output: Inicializovana struktura *key_chain*
    key_chain ← new_key_chain_structure()
    key_chain.master_key ← *master_key* ▷ Hlavny kluc
    key_chain.epoch ← 0 ▷ Zaciatok od epochy 0
    key_chain.is_initiator ← *is_initiator*
    key_chain.auth_key_curr ← derive_authentication_key(*master_key*)
    if *is_initiator* **then** ▷ Klient pripravuje kluce
        temp_master ← copy(*master_key*)
        temp_auth ← copy(*key_chain.auth_key_curr*)
        evolve_keys(*temp_master*, *temp_auth*, 1) ▷ Pre epochu 1
        key_chain.auth_key_next ← *temp_auth*
        key_chain.auth_key_prev ← *key_chain.auth_key_curr*
        secure_wipe(*temp_master*) ▷ Bezpecne vymazanie docasneho kluca
    else ▷ Server inicializuje vsetky kluce rovnako
        key_chain.auth_key_prev ← *key_chain.auth_key_curr*
        key_chain.auth_key_next ← *key_chain.auth_key_curr*
    end if
    return *key_chain*
end function
function DERIVE_AUTHENTICATION_KEY(*master_key*)
    Input: *master_key* (32B kluc)
    Output: *auth_key* (32B autentifikacny kluc)
    auth_key ← crypto_blake2b(*master_key* ‖ "SAKE_K_AUTH", 32) ▷ Hashovanie s oddelenou domenou
    return *auth_key*
end function

Algorithm 2 SAKE: Proces vzajomnej autentifikacie

```
1: function AUTHENTICATE_SESSION(key_chain, is_initiator)
2:     Input: struktura key_chain, priznak is_initiator
3:     Output: session_key alebo chybovy kod
4:     if is_initiator then                                              ▷ Strana klienta
5:         client_nonce ← generate_random_bytes(16)
6:         send_all(socket, client_nonce, 16)                            ▷ Odoslanie nonce serveru
7:         server_nonce ← recv_all(socket, 16)                           ▷ Prijatie nonce servera
8:         challenge ← recv_all(socket, 32)                              ▷ Prijatie vyzvy
9:         response ← compute_response(key_chain.auth_key_curr, challenge, server_nonce)
10:        send_all(socket, response, 32)                                ▷ Odoslanie odpovede serveru
11:    else                                                              ▷ Strana servera
12:        client_nonce ← recv_all(socket, 16)                           ▷ Prijatie nonce klienta
13:        server_nonce ← generate_random_bytes(16)
14:        challenge ← generate_challenge(key_chain.auth_key_curr, client_nonce, server_nonce)
15:        send_all(socket, server_nonce, 16)                            ▷ Odoslanie nonce servera
16:        send_all(socket, challenge, 32)                               ▷ Odoslanie vyzvy
17:        client_response ← recv_all(socket, 32)                        ▷ Prijatie odpovede klienta
18:        expected_response ← compute_response(key_chain.auth_key_curr, challenge, server_nonce)
19:        if client_response ≠ expected_response then
20:            return ERROR_AUTHENTICATION_FAILED                        ▷ Autentifikacia zlyhala
21:        end if
22:    end if
23:    session_key ← derive_session_key(key_chain.master_key, client_nonce, server_nonce)
24:    update_key_chain(key_chain)                                       ▷ Vyvoj klucov
25:    return session_key
26: end function
```

Algorithm 3 SAKE: Pomocne funkcie autentifikacie

```
1: function GENERATE_CHALLENGE(auth_key, client_nonce, server_nonce)
2:     Input: Autentifikacny kluc, nonce klienta, nonce servera
3:     Output: Vyzva vyzadujuca znalost auth_key
4:     challenge ← crypto_blake2b(auth_key ‖ client_nonce ‖ server_nonce ‖ "SAKE_CHALLENGE", 32)
5:     return challenge
6: end function
7: function COMPUTE_RESPONSE(auth_key, challenge, server_nonce)
8:     Input: Autentifikacny kluc, vyzva, nonce servera
9:     Output: Odpoved preukazujuca znalost auth_key
10:    response ← crypto_blake2b(auth_key ‖ challenge ‖ server_nonce, 32)
11:    return response
12: end function
13: function DERIVE_SESSION_KEY(master_key, client_nonce, server_nonce)
14:    Input: Hlavny kluc, nonce klienta, nonce servera
15:    Output: Relacny kluc pre bezpecnu komunikaciu
16:    session_key ← crypto_blake2b(master_key ‖ client_nonce ‖ server_nonce ‖ "SAKE_SESSION", 32)
17:    return session_key
18: end function
```

Algorithm 4 SAKE: Postup aktualizacie retaze klucov

function UPDATE_KEY_CHAIN(*key_chain*)
    Input: struktura *key_chain* na aktualizaciu
    Output: Aktualizovana *key_chain* s vyvinutymi klucmi
    if *key_chain.is_initiator* **then** ▷ Iniciator (klient)
        key_chain.auth_key_prev ← *key_chain.auth_key_curr* ▷ Rotacia klucov
        key_chain.auth_key_curr ← *key_chain.auth_key_next*
        temp_master ← copy(*key_chain.master_key*)
        next_epoch ← *key_chain.epoch* + 1
        temp_master ← crypto_blake2b(*key_chain.master_key* ‖ *next_epoch* ‖ "SAKE_K", 32)
        key_chain.auth_key_next ← derive_authentication_key(*temp_master*)
        key_chain.master_key ← crypto_blake2b(*key_chain.master_key* ‖ *key_chain.epoch* ‖ "SAKE_K", 32)
        secure_wipe(*temp_master*) ▷ Bezpecne vymazanie docasneho kluca
    else ▷ Odpovedajuci (server)
        key_chain.master_key ← crypto_blake2b(*key_chain.master_key* ‖ *key_chain.epoch* ‖ "SAKE_K", 32)
        key_chain.auth_key_curr ← derive_authentication_key(*key_chain.master_key*)
        key_chain.auth_key_prev ← *key_chain.auth_key_curr*
        key_chain.auth_key_next ← *key_chain.auth_key_curr*
    end if
    key_chain.epoch ← *key_chain.epoch* + 1 ▷ Zvysenie epochy
end function