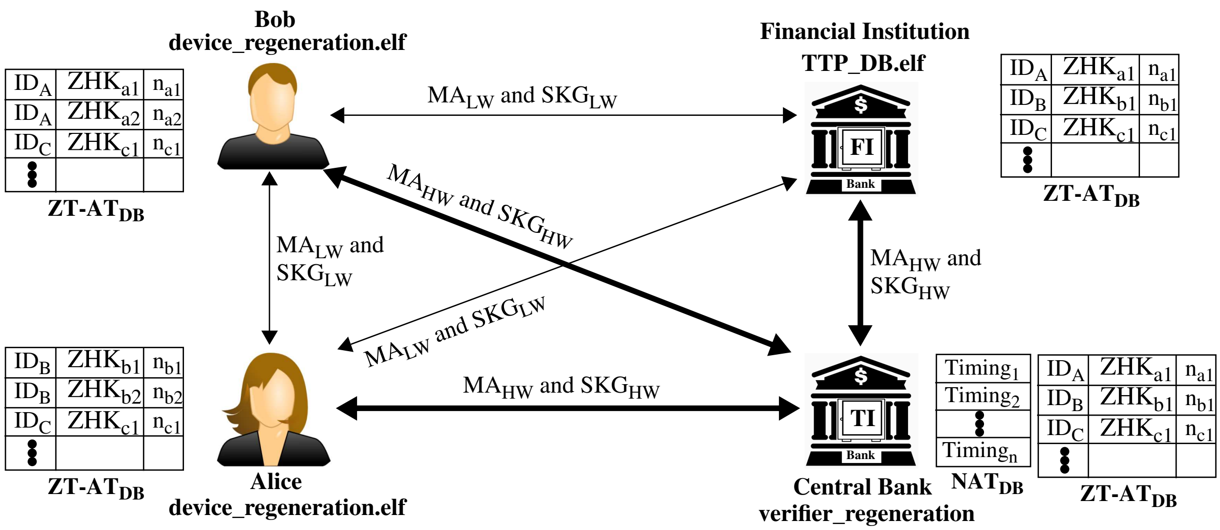
LAB Assignment #7 for ECE 525

Description: ZeroTrust Light-Weight Authentication

The overall setup for PUF-Cash is shown in the following figure:

FI: Financial Institution	AT: Authentication Token	MA_{LW}: Light-weight Mutual Authentication
TI: Token Issuer (Central Bank)	HW: Heavy-weight (Timing DB)	SKG_{LW}: Light-weight Session Key Generation
	LW: Light-weight (AT-based)	MA_{HW}: Heavy-weight Mutual Authentication
		SKG_{HW}: Heavy-weight Session Key Generation



In previous labs, PUF authentication and key generation was done between devices, e.g., Alice (Bob), and a secure server, e.g., a Bank, using the timing database created during provisioning (shown as thick black lines in figure).

We classify this type of authentication as 'heavy weight' (HW) because it utilizes a large database on the server to 'clone' a portion of the challenge-response-pairs associated with the devices (see Appendix).

As an alternative, PUF-Cash utilizes a light-weight version of the timing database to enable devices to authenticate between themselves (and with the financial institution), and without the need for a trusted authority, e.g., the Central Bank or Token Issuer or TI, to be involved.

The light-weight version utilizes authentication tokens (ATs) that are constructed using the SiRF-PUF on each device. As is true of all authentications, **the AT can only be used once in an authentication operation**, and therefore, the AT scheme requires a refresh operation to replace the 'used' AT with a fresh one..

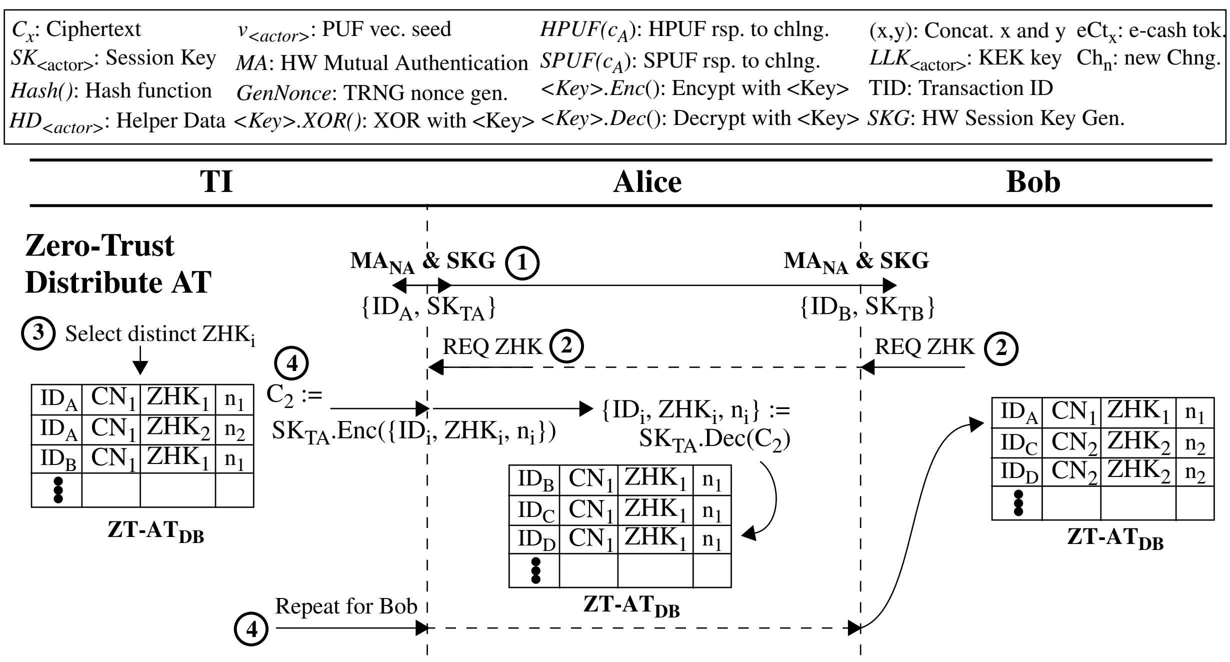
In previous labs, we ran code that carried out heavy-weight (HW) authentication, which utilizes the timing databases stored at the Central Bank (also called the TI or token issuer).

We also added code to allow Alice and Bob to generate AT for light-weight authentication. The AT were transmitted to the TI for storage in a database.

In this lab, we add code which allows Alice and Bob to request unique subsets of ATs from the TI that are associated with other party(s). We refer to this as distributing the ATs. The AT are used for in-field authentication and session key generation as shown by the following diagram.

This function is invoked by selecting Option 8 from the menu after you run device_regeneration.elf on Alice and Bob's FPGAs.

The message exchange diagram associated with the distribute AT operation is shown below.



- Step 1) Alice/Bob and TI authenticate and generate a session key using the heavy weight functions.
- Step 2) Alice/Bob request ZHK (AT) from the TI.
- Step 3) TI selects a unique subset of ZHK from it's master database, ZT-AT_{DB}, one for each customer excluding self.
- Step 4) TI encrypts the entries and transmits them to Alice and Bob. Alice and Bob decrypt and store them in its local ZT-AT_{DBc}.

In-Field Zero-Trust Authentication and Session Key Generation:
In this lab, we also add code for the ZeroTrust protocol between Alice and Bob.

The ZT-AT in-field process is carried out when Alice contacts Bob to pay for goods or services in an environment where connectivity exists only between Alice and Bob (no TI is available). The message exchange protocol is shown in the following figure, and is described as a sequence of the following operations:

The transaction begins with Alice sending Bob a request to authenticate and generate a shared session key.

- Step 1) Alice requests lightweight authentication and session key generation to Bob
- Step 2) Alice sends Bob an identifier, ID_A, that allows Bob to locate the corresponding AT in his $ZT\text{-}AT_{DB}$.
- Step 3) Bob responds to Alice with an Ack or Nak (as status) on whether or not he possesses an AT for Alice. He also responds with his own identifier ID_B in cases where he possesses an AT for Alice.
- Step 4) Alice determines if she has an AT for Bob in her $ZT\text{-}AT_{DB}$ using the Bob's ID_B.
- Step 5) She transmits a corresponding Ack or Nak to Bob.
- Step 6) Assuming both Alice and Bob have AT for each other, they both retrieve the AT for the other party from their $ZT\text{-}AT_{DB}$, which is represented by the tuple $\{ZHK_x, n_x\}$ with x := b or a, respectively.

Shared Key Generation:

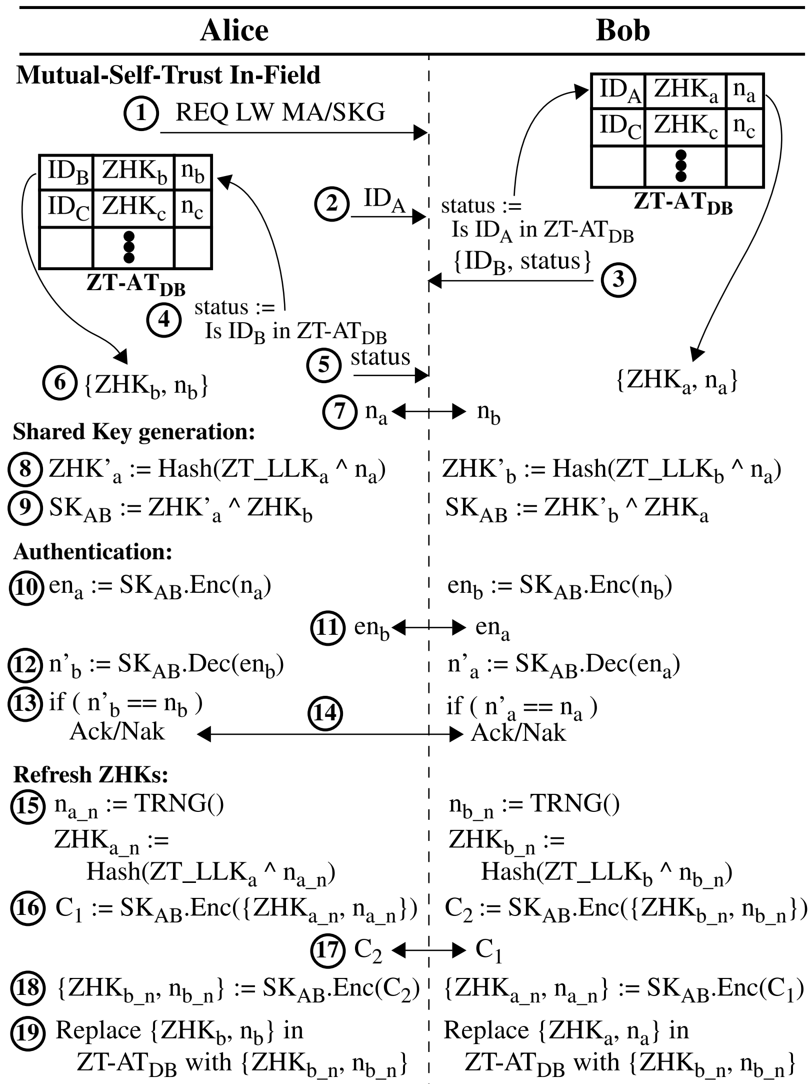
- Step 7) Alice and Bob exchange the nonce components, n_x, of the AT.
- Step 8) Both parties regenerate their long-lived keys ZT_LLK using challenge information stored in their LLK_{DB}, and then compute a local version of the ZHK'_x using $Hash(ZT\_LLK_x$ XOR $n_x)$ (NOTE: XOR operation is annotated as ^ in the diagram).
- Step 9) Alice and Bob create a shared key SK_{AB} by XOR'ing the local copy of ZHK'_x with the ZHK_x that they store for the other party in their $ZT\text{-}AT_{DB}$.

Authentication:

- Step 10) Authentication begins with Alice and Bob encrypting the n_x they received from the other party with the newly created shared key SK_{AB} to create en_x.
- Step 11) Alice and Bob exchange the encrypted nonces en_x.
- Step 12) Alice and Bob decrypt the en_x using the shared key.
- Step 13) Alice and Bob compare the decrypted n_x with the ones they store in their $ZT\text{-}AT_{DB}$.
- Step 14) The status of the comparison is shared with the other party with each transmitting an Ack or Nak.
  Alice and Bob have authenticated and posses a shared key at this point assuming both have acknowledged that the nonces n_x match their own local copies.

A refresh operation is carried out in Steps 15) through 19), which is left as an exercise. This allows Alice and Bob to carry out another, future, transaction without returning the TI to get new AT.

C_x: Ciphertext	$v_{<actor>}$: PUF vec. seed	$HPUF(c_A)$: HPUF rsp. to chlng.	(x,y): Concat. x and y	eCt_x: e-cash tok.
$SK_{<actor>}$: Session Key	*MA*: HW Mutual Authentication	$SPUF(c_A)$: SPUF rsp. to chlng.	$LLK_{<actor>}$: KEK key	Ch_n: new Chng.
Hash(): Hash function	*GenNonce*: TRNG nonce gen.	<Key>.*Enc*(): Encypt with <Key>	TID: Transaction ID	
$HD_{<actor>}$: Helper Data	<Key>.*XOR()*: XOR with <Key>	<Key>.*Dec*(): Decrypt with <Key>	*SKG*: HW Session Key Gen.	



REMEMBER, once the original AT is used, it MUST BE DISCARDED to avoid replay attacks.

This function can be invoked by pressing Option 2. Option 2 is labeled Transfer, which you will expand on in the project to allow Alice to pay Bob.

The following describes the sequence of operations that occurs in device_regeneration.elf, which Alice and Bob run on the FPGA.

- GenLLK(): Generate a long-lived key (LLK) with the TI, which Alice, Bob, FI will use to generate AT.

if LLK exists
  Regenerate LLK with SiRF
else
  MA_{HW}, SKG_{HW} with TI
  Get Chlng
  Generate LLK with SiRF
  Store Chlng info to LLK Table

PUF-Cash DB
LLK Table

ID	AID	mask	Chlng	status

ID: chip #
AID: anonymous chip #
mask: Components of Chlng
Chlng: vectors, params, etc
status: 0: un-used, 1: used

- ZeroTrustGetCustomerATs()

if AT do NOT exist
  ZeroTrust_Enroll()

ZeroTrust_Enroll()
  If LLK non-null, ERROR
  MA_{HW}, SKG_{HW} with TI
  Get number of AT to generate from TI
  For each AT
    Generate nonce, n_x
    CH_LLK = hash(LLK XOR n_x)
    encrypt(CH_LLK) and send to TI
    encrypt(n_x) and send to TI
    TI adds to ZeroTrustAuthenToken table

AuthenticationToken DB
ZeroTrustAuthenToken Table

ID	CH_LLK	n_x	status

ID: chip #
AID: anonymous chip #
CH_LLK: hash(LLK XOR n_x)
n_x: nonce
status: 0: un-used, 1: used

- AliceGetClient_IPs()

MA_{HW}, SKG_{HW} with TI
Get customer IPs from TI
Get TTP IP from TI
Store results Client_CIArr data structure

- Loop forever:

    Set up socket to listen for connections from Alice or Bob
    If Alice or Bob requests connection
      ProcessInComingRequest()
    Get user request from Alice or Bob
    If MENU_TRANSFER
      AliceTransferDriver()
    If MENU_GET_AT
      ZeroTrust_GetATs()

ZeroTrust_GetATs()
  MA_{HW}, SKG_{HW} with TI
  Alice gets ATs for Bob, Bob gets ATs for Alice, stores them in their
    ZeroTrustAuthenToken Table

AliceTransferDriver()
  Open socket to Bob
  AliceDoZeroTrust()
    ExchangeIDsConfirmATExists()
      Send Alice ID to Bob
      Bob checks if he has Alice AT (ZeroTrustGetCustomerATs())
      Alice gets status from Bob and Bob ID
      Alice check if she has Bob AT (ZeroTrustGetCustomerATs())
      If both Alice and Bob have ATs for each other
        ZeroTrustGenSharedKey()