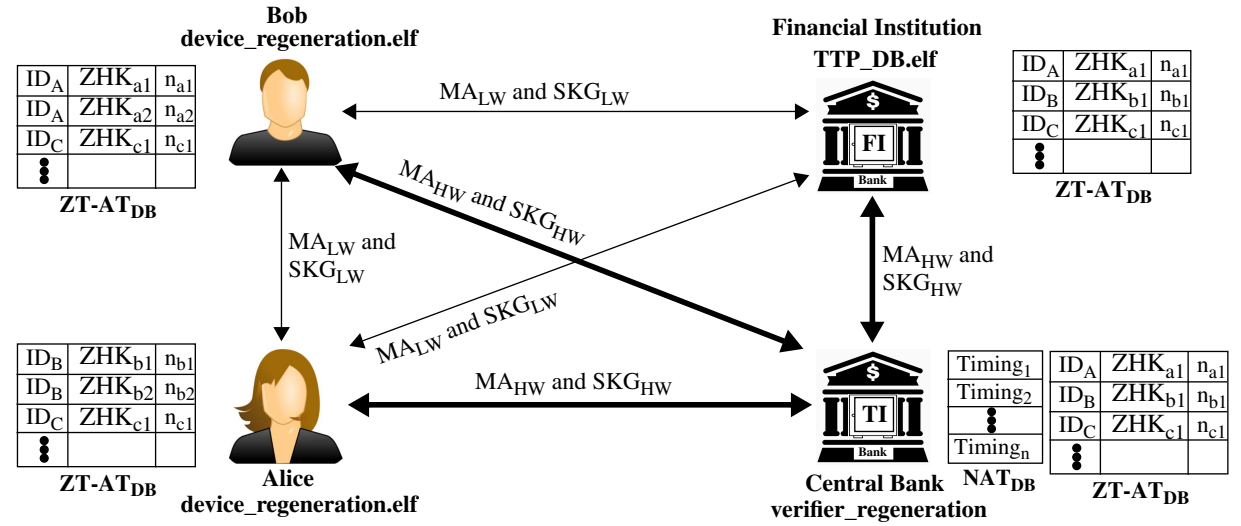
LAB Assignment #6 for ECE 525

Description: ZeroTrust Light-Weight Authentication, Generate and Distribute ATs

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

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



Alice, Bob are single-threaded and run on ZYBO and/or CORA FPGAs (**device_regeneration.elf**)
FI is multi-threaded and runs on ZYBO or CORA **(TTP_DB.elf**)
TI (central bank) is multi-threaded and runs on a laptop (**verifier_regeneration)**

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).

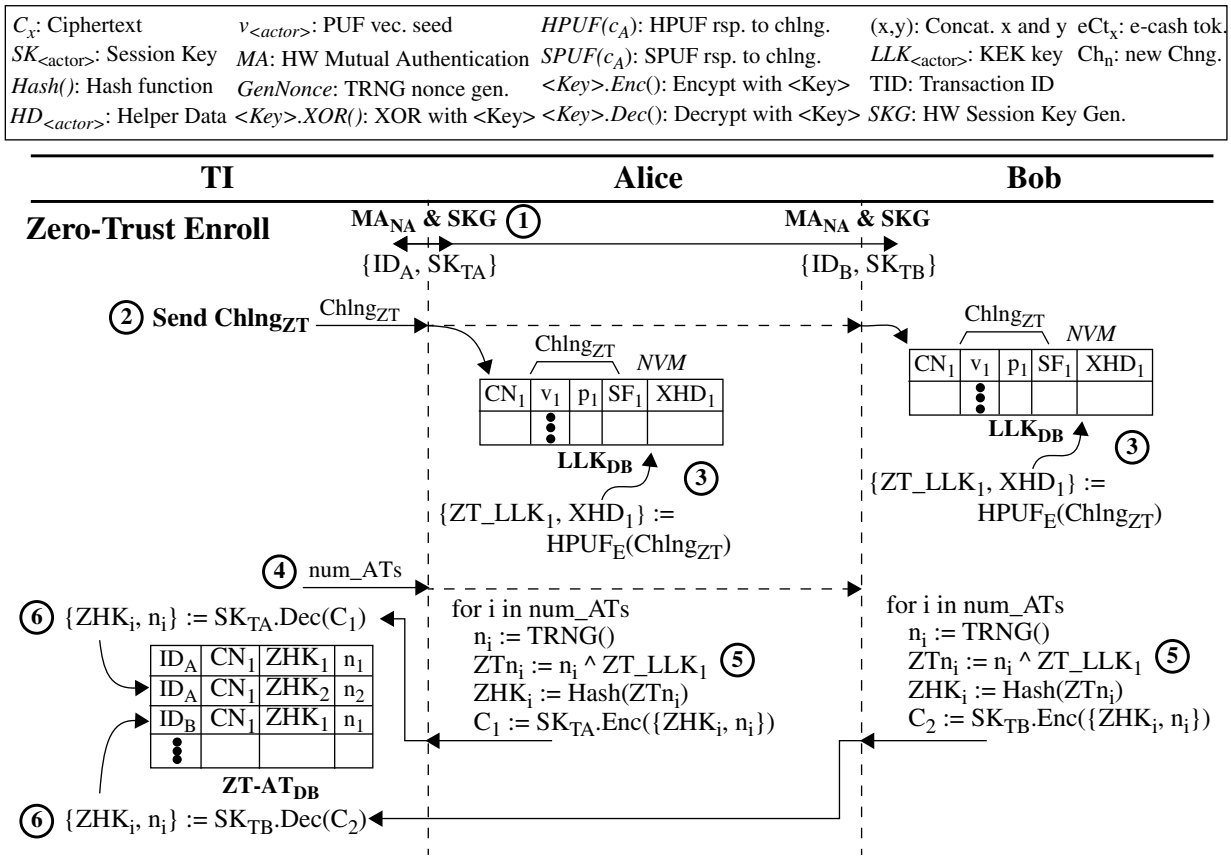
We classify this type of authentication as 'heavy weight' (HW) because it utilizes a large database on the server to 'clone' a portion of the challenge-response-pairs associated with the devices (see Appendix).

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 this lab, we do the first part of the Zero-Trust light-weight authentication and session key generation protocol.

The message exchange diagram for creating the light-weight ATs is shown in the following figure. We refer to this model of authentication and session key generation as zero-trust, because Alice and Bob establish a secure communication channel without utilizing the TI.



The AT are created by running the PUF in long-lived key (LLK) generation mode. LLK mode refers to the ability of the SiRF PUF to generate (enrollment) and then re-generate (regeneration) the same key later in time using helper data generated during enrollment.

The enrollment process requires the TI to be involved. For each AT to be generated, the following operations are carried out:

- Step 1) Alice and TI authenticate and generate a session key using the heavy weight functions
- Step 2) The TI sends at least one challenge to the device, which its stores in its LLK_{DB}
- Step 3) The device runs the SiRF PUF in LLK mode to generate helper data, XHD, which is stored in the LLK_{DB}
- Step 4) The TI sends a request to the device to generate *p* ATs
- Step 5) For each AT, the device runs it SiRF TRNG to generate a nonce, n_i. The device XORs the n_i with the ZT_LLK, and then hashes the XOR'ed bitstring to create an ZHK, i.e., ZHK := (nonce XOR ZT_LLK).

The XOR operation cryptographically binds the nonce to the ZT_LLK, while the hash operation obfuscates the ZT_LLK, i.e, the ZT_LLK cannot be recovered from the hash because the hash is a one-way function.
This allows the device to openly transmit the AT, i.e., nonce plus ZHK, over an insecure network while preventing adversaries from snooping and collecting model-building data about a device.

- Step 6) The device encrypts the AT using the HW session key from Step 1, and transmits the encrypted AT to the TI. The TI stores the AT in the ZT-AT_{DB}.

Note that the device does NOT store the ZT_LLK or ZHK, only the challenge and helper data needed to reproduce them.

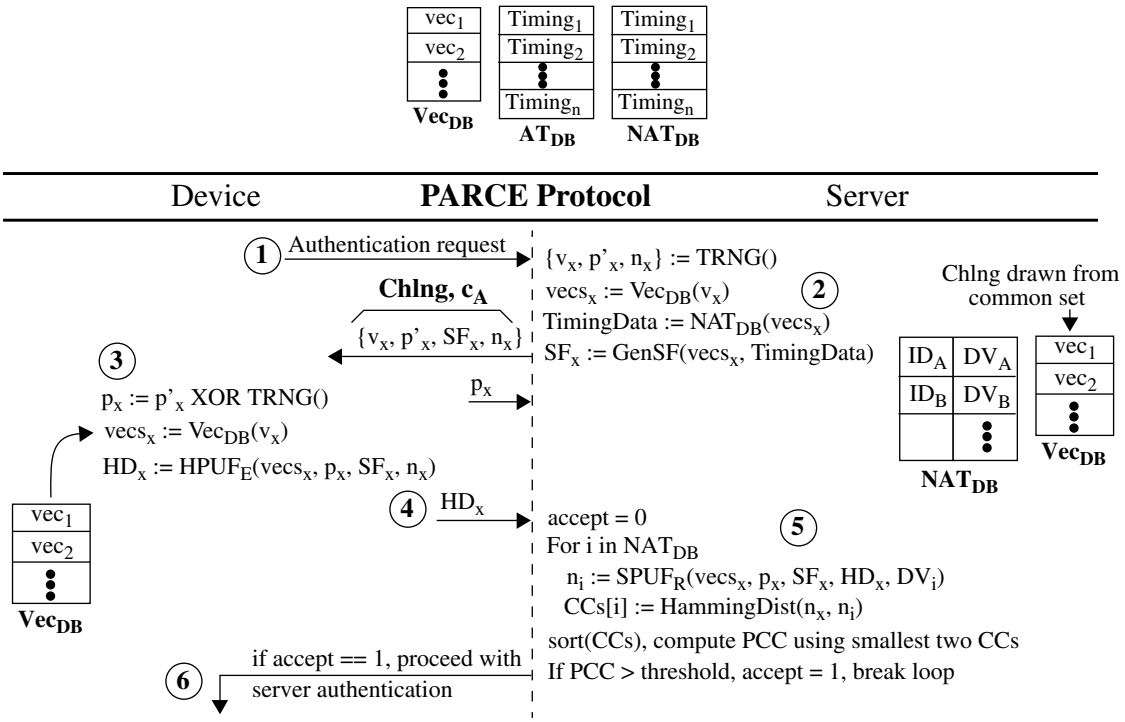
Later in the field, it regenerates the ZT_LLK and carries out the same sequence to reproduce the ZHK as needed

The TI collects multiple ZHK from all devices in the population and stores them in a master ZT-ZT-AT_{DB}.

Appendix: Heavy-Weight Authentication

The message exchange diagram for heavy-weight (HW) functions is shown in the figure.

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.	



The following details the operations performed:

- The device requests authentication from a server.
- The server generates a random vector-selection seed v_x, random nonces n_x and p'_x, and a set of SpreadFactors, SF_x, collectively referred to as the Chlng, c_A. The seed v_x will be used by the server and device to select a set of vectors from the common pool of vectors stored in the $Vecs_{DB}$. The nonce n_x represents the authentication nonce. The nonce p'_x is used to specify parameters to the SiRF algorithm once it is XORed with a corresponding device-generated nonce. The SF_x are derived from DV_i stored in the NAT_{DB} and are used to increase the number of strong (unable) bits produced by the SiRF PUF. The v_x, p'_x, SF_x and n_x are transmitted to the device.
- The device XORs p'_x with its own TRNG-generated version to produce p_x, which is transmitted back to the server. The p_x construction and exchange process prevents adversaries from engaging in chosen-message attacks on the device. The device extracts $vecs_x$ from its Vec_{DB} and applies $vecs_x$, p_x, SF_x and n_x to its hardware PUF, $HPUF_E$, in enrollment mode. The PUF produces only a helper data bitstring, HD_x, when configured in SKE mode.
- The device transmits HD_x to the server.
- The server performs authentication by searching its NAT_{DB} for a device *i* that can reproduce n_x. For each device *i* in the NAT_{DB}, the DV_i corresponding to the vectors $vecs_x$, along with p_x,

SF_x and the device-generated HD_x, are used as input to a software version of the PUF, $SPUF_R$, configured to run in regeneration mode. A correlation coefficient (CC) is computed for each device in the database using n_i and n_x as input to a function called HammingDist.

- HammingDist counts the number of mismatching bits that exist between n_x and n_i. The CC array is sorted once all CCs are computed, and then the two smallest CCs are used to compute a percentage-change coefficient, PCC, which represents the percentage difference between to two smallest PCCs. If the PCC exceeds a server-defined threshold, the server accepts the authentication request, otherwise it rejects and halts the process.

Server Authentication:
In cases where the server accepts the device authentication request, the device performs a similar process to authenticate the server. Although the process is NOT shown in the figure, the following operations are carried out. The device generates a nonce n_y, and transmits it to the server. The server generates a new challenge and runs $SPUF_E$ (enrollment) using the DV_i from its NAT_{DB} (it knows the identity of the device) to produce a helper data bitstring HD_y. The challenge and HD_y are transmitted to the device. The device runs the SiRF PUF in regeneration mode, $HPUF_R$, to generate n'_y and computes a CC using the HammingDist function. The device accepts the authentication if the CC is less than a threshold, otherwise it rejects. The server is notified of the authentication decision and proceeds to engage with the device to generate a shared key if the device accepted the authentication request.

Although the NAT_{DB} provides an exponential number of CRPs for each device, it is large (multiple GB with 10's of thousands of devices) and therefore, not suitable for storage on resource-constrained devices. Moreover, the NAT_{DB} holds many secrets for all fielded devices and even if it could be stored, it would not be secure to distribute the NAT_{DB} to all devices.