OAuth 2.0-based authentication solution for FPGA-enabled cloud computing

Semih Ince, David Espes, Guy Gogniat, Julien Lallet, Renaud Santoro

▶ To cite this version:

Semih Ince, David Espes, Guy Gogniat, Julien Lallet, Renaud Santoro. OAuth 2.0-based authentication solution for FPGA-enabled cloud computing. 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion, Dec 2021, Leicester, United Kingdom. 10.1145/3492323.3495635 . hal-03606350

HAL Id: hal-03606350

https://hal.science/hal-03606350

Submitted on 11 Mar 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

OAuth 2.0-based authentication solution for FPGA-enabled cloud computing

Semih Ince
semih.ince@nokia.com
Nokia Bell Labs

David Espes
david.espes@univ-brest.fr
Univ. Bretagne Occidentale

Guy Gogniat
guy.gogniat@univ-ubs.fr
Univ. Bretagne Sud

Julien Lallet
julien.lallet@nokia.com
Nokia Bell Labs

Renaud Santoro
renaud.santoro@nokia.com
Nokia Bell Labs

ABSTRACT

FPGA-enabled cloud computing is getting more and more common as cloud providers offer hardware accelerated solutions. In this context, clients need confidential remote computing. However Intellectual Properties and data are being used and communicated. So current security models require the client to trust the cloud provider blindly by disclosing sensitive information. In addition, the lack of strong authentication and access control mechanisms, for both the client and the provided FPGA in current solutions, is a major security drawback. To enhance security measures and privacy between the client, the cloud provider and the FPGA, an additional entity needs to be introduced: the trusted authority. Its role is to authenticate the client-FPGA pair and isolate them from the cloud provider. With our novel OAuth 2.0-based access delegation solution for FPGA-accelerated clouds, a remote confidential FPGA environment with a token-based access can be created for the client. Our solution allows to manage and securely allocate heterogeneous resource pools with enhanced privacy & confidentiality for the client. Our formal analysis shows that our protocol adds a very small latency which is suitable for real-time application.

CCS CONCEPTS

• **Security and privacy** → **Authorization**; **Access control**; **Privacy-preserving protocols**; *Multi-factor authentication*; • **Networks** → **Cloud computing**; • **Hardware** → **Hardware accelerators**; *Reconfigurable logic applications.*

KEYWORDS

Cloud, FPGA, security, authentication, OAuth

ACM Reference Format:
Semih Ince, David Espes, Guy Gogniat, Julien Lallet, and Renaud Santoro. 2021. OAuth 2.0-based authentication solution for FPGA-enabled cloud computing. In *2021 IEEE/ACM 14th International Conference on Utility and Cloud Computing (UCC '21) Companion (UCC '21 Companion), December 6–9, 2021, Leicester, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3492323.3495635

1 INTRODUCTION

FPGAs are more and more popular in cloud computing as accelerators thanks to their high flexibility and performance. Due to their inherent nature, FPGAs can deal with high computational load and are highly interesting regarding performance and power consumption compared to General Purpose Processors [1] or GPUs. FPGAs are highly investigated for applications requiring intensive computations (e.g., fully homomorphic encryption algorithm). Furthermore, multi-tenancy enables efficiency and flexibility in a cloud usage. In FPGA, a multi-tenancy context allows multiple users to share the single FPGA component, introducing new security challenges.

Cloud security is critical for a client when choosing a commercial Cloud Provider (CP). Commercial cloud users expect secure remote computation and access to FPGA accelerators with minimal impact on their design performance. Security mechanisms need to be adapted for an appropriate cloud usage. First, the client needs to ensure that its data is kept private. The client does not want to disclose sensitive Intellectual Property (IP) and data to the cloud provider. To ensure that, the client needs an encrypted channel with the FPGA isolated from the CP. Furthermore, authentication is another important security aspect to establish secure remote connection between a client and the hardware acceleration material. The client needs to ensure that the correct FPGA is used and that no other users may access the allocated resources. Authentication is necessary to manage FPGAs and different cloud service accesses to mitigate client impersonations and data breaches.

Methods used by different CPs lack of transparency concerning data encryption methods, bitstream protection and IP theft. To remove this drawback, it is necessary to use methods and protocols which respect user privacy and intellectual property. A solution to reinforce these aspects is to introduce an intermediate authority between the client and the CP. This authority would be similar to already existing entities in the Public Key Infrastructure mechanism (e.g., certificate authority). Thus, we need an entity that the CP and the client can trust so they do not have to trust each other. The Trusted Authority (TA) serves this purpose in our proposal. Often, the TA would implicitly be the chip manufacturer for practical and security reasons. It can safely implement security mechanisms and store shared secrets inside FPGAs in the production phase. The chip manufacturer is responsible of the FPGA manufacturing security. It ensures that FPGAs and implemented security primitives are not tampered or stolen. From a client's perspective, the TA achieves device authentication and isolation from the CP by using the shared

secret inside the FPGA. Thus, the client can protect its sensitive IP and data from the CP. From the CP's perspective, the TA achieves tasks like FPGA access management and authentication.

OAuth 2.0 is a secure access delegation open standard where a resource owner can share resources with a client thanks to a common trust placed in a third party (i.e., the TA) [2]. Our solution adapts this protocol for a cloud-enabled FPGA context.

This paper tackles the four-entity authentication problem (client, FPGA, TA, and CP) with a novel OAuth 2.0-based access delegation solution for FPGA-accelerated clouds.

The contributions of this paper are described as follows:

- A new OAuth 2.0-based authentication solution for cloud enabled FPGA.
- Authentication method for 4 entities: TA, CP, FPGA and client.
  - Adapted to cloud usage to ensure isolation between the client and the CP.
  - Low latency single sign-on tokenized FPGA access.
- Client's design and data isolated from CP.
  - Token-based FPGA access for the client produced with a shared secret between the TA and the FPGA.

This paper is organized as follows. The state of the art of Cloud FPGA security and mechanisms is introduced in Section 2. Then, the proposed OAuth2.0-based authentication solution is described in Section 3. A performance analysis of the presented solution is provided in Section 4.

2 RELATED WORKS

2.1 Trusted authority for cloud-enabled FPGA

Some cloud virtualization or attestation solutions like [3], [4], [5] do not make use of a trusted authority. In [3], authors propose a self attestation mechanism for FPGA devices. Without needing a trusted third party, the hardware and software stacks inside the FPGA device are proven tamper resistant thanks to a prover-verifier relation between an FPGA and an external verifier. However client privacy and security are not optimal, the CP has the verifier role and has access to security primitives inside the FPGA. In [4] authors propose a unified FPGA resource pool with virtualization methods to achieve high abstraction and flexibility for end users. This implies that a user needs to disclose its source code to the cloud provider to generate multiple bitstreams for every FPGA region. This is necessary because a bitstream is generated for a targeted partially reconfigurable FPGA region. This brings privacy, security and IP theft concerns for the client and a trusted authority could be one solution to solve these issues. The authors of [5] propose a virtualized execution runtime for cloud FPGAs. Even though the goal of their paper is not security oriented, the potential end user of the proposed framework is dependent of the cloud provider's tools and APIs. Some functions like bitstream verification and access management could be migrated to a trusted authority to achieve greater privacy and security for the client.

To enable FPGAs in cloud computing, several authors agreed on the necessity of a trusted authority different from the cloud provider [6], [7], [8], [9], [10]. For practical reasons, the FPGA manufacturer is usually considered as the trusted authority and has an important role in the security of the remote connection between the client and the FPGA. By managing and securing the production process of their FPGAs, the manufacturer can securely implement unique keys storage and/or Physical Unclonable Function (PUF). That makes the FPGA manufacturer a legitimate candidate for being a trusted authority in cloud computing. Also, security sensitive services like bitstream verification and FPGA authentication can be delegated to the TA to achieve isolation between the client and the cloud provider. Currently, AWS and Microsoft Azure propose bitstream verification, but both their respective methods have privacy concerns on user bitstream. In fact, the client needs to disclose its bitstream or netlist to the CP for security purposes using the provided scripts. Few design patterns like combinatorial loops are prohibited because they lead to security problems in a cloud usage. Even if the data is encrypted during transport, it is decrypted on the CP's side which is not optimal.

2.2 Security of cloud-enabled FPGA

In [6] and [7], secure FPGA enclaves are proposed. A TA is present in the communication scheme alongside the CP and the client, but their approach and achievements differ. Both solutions, and [10] agree to use a PUF or stored keys inside an FPGA. Authors in [9] use stored key values for symmetric encryption of client bitstream and data, isolating the client from the CP.

In [6] and [9], the FPGA device is bootstrapped and introduced into a Public Key Infrastructure (PKI). In [6] device unique keys (e.g., PUF) are derived to constitute a key hierarchy and obtain key-pairs for bitstream encryption and enclave communication/identification. Unlike [7], the solution for FPGA authentication proposed here is an SGX-inspired attestation mechanism endorsed by the TA. The latter offers services like bitstream certification and boot code authentication. To provide a secure FPGA environment, security critical components like the device unique key and bitstream loader are controlled by the TA. This way, the TA controls the root of the key hierarchy and can update/revoke keys depending on security threats and system update. Thus, the CP is much more isolated from the client design.

The authors of [7] propose a framework where FPGA authentication is achieved and a secure channel is created for the client using a modular exponentiation and Diffie-Hellman Ephemeral algorithm. The client compares the FPGA PUF output with the response known by the TA. If valid, the client now has a shared secret with the FPGA and the session key is established. Thus, client bitstream and data will be protected against the CP and bitstream integrity will be achieved. But unlike [6], multi-tenancy is not enabled, and this represents a major drawback for dynamic and flexible cloud usage.

None of the previously cited works proposes a client authentication mechanism. Existing communication and authentication schemes lack a client-side authentication. In [7], a user requests access to an FPGA from the CP and obtains in return the FPGA serial number. The client sends it to the TA and proceeds to FPGA authentication. A malicious user can impersonate a client and get access to the provided FPGA before the client. In fact, the client obtains access to the FPGA after the session key is set up, but the client is not authenticated. This means the session key is security critical and must be kept secret. The whole security of the remote access and computing lies on the session key. Attack vectors for

this use case are well known (i.e., man in the middle, client-side malware...).

If this client authentication process is ignored, malicious users could have a backdoor access to assigned FPGAs and this could lead to IP theft and data leak. In fact client authentication is a security critical process as it is the base of the remote and secure computation. Every other security mechanisms coming after a missing authentication process would be useless.

3 NEW THIRD PARTY AUTHENTICATION MECHANISM FOR CLOUD ENABLED MULTI-TENANT FPGA

3.1 Problem Statement

3.1.1 *The absence of a trusted authority.*

Without a TA, the client accesses the FPGA with tools provided by the CP (most frequently a virtual machine). The CP has every right on its resource. Bitstream and data can be recovered by the CP because disclosing the bitstream to the CP is mandatory for security checks on user program. The encryption keys or certificates could be generated by the CP for the FPGA so the connection would not be private between the client and its allocated resources.

By introducing a trusted authority that both the client and the cloud provider can trust, it is possible to create a secure remote computation for the client and meet the cloud provider's security requirements at the same time. With four entities in the authentication scheme, the client can use the FPGA owned by the cloud provider without disclosing sensitive data.

The cloud provider does not lose the control of its devices. They can still be managed, and clients' requests can be accepted or declined. The cloud provider still has an important role in the allocation of FPGAs. Instead of having a single entity controlling the FPGA (i.e., CP), two entities (i.e., CP and TA) have rights over the allocated resource. Now the CP does not have full privilege over the allocated resource, some of its operations are delegated to the TA. As mentioned in Section 1, the TA should be the chip manufacturer. Security elements can be privately implemented without giving the CP access to them. By splitting the roles and responsibilities, we ensure that one entity does not own all the privileges of an FPGA.

3.1.2 *Lack of FPGA user authentication.*

Additionally, in existing solutions, only the client authenticates with the cloud provider. To establish a trust relation, the trusted authority needs to be introduced to the client and have a way to authenticate him without requiring the cloud provider's services. The client needs a transparent authentication scheme with the trusted authority to establish the basis of the secure and remote FPGA access. In current FPGA cloud solutions, clients use virtual machines to access their resources. There are no other security measures to protect the resource. Thus, a compromised virtual machine can lead to malicious behavior and client impersonation.

With the introduction of a TA, we plan to solve problems like user privacy, user and hardware authentication with third party implication, and create a private channel between the FPGA and the client, isolated from the CP and the TA. Finally, bypassing current tools like virtual machines and offering a direct secure client-FPGA channel ensures privacy and data protection.

3.2 Access delegation open standard: OAuth 2.0

OAuth 2.0 is an open standard authorization framework which enables a third party to get limited access to an HTTP service on behalf of the Resource Owner (RO) [2]. Often it is used by companies such as Amazon and Google to share user information with third party websites or applications.

This framework works with four different entities: an Authorization Server (AS), a Resource Owner (RO), a Resource Server (RS) and a client. A use case of OAuth 2.0 could be a third party website allowing visitors to register themselves using another website's account information. The third party website (client) would need the user's permission (resource owner) to get access to the personal information stored in the other website's servers (resource server).

The high level protocol flow is described in Fig. 1.

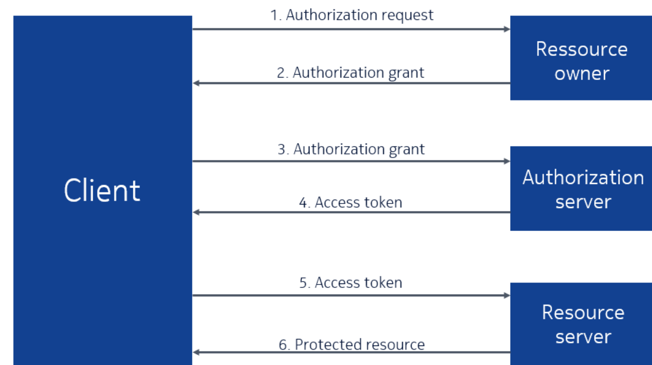

Figure 1: OAuth 2.0 high level execution flow

(1) The client requests a resource from the RO through its interface (e.g., website)
(2) According to the grant scheme employed, the RO issues (if he accepts the request) an authorization grant.
(3) The client authenticates himself with the AS and uses the authorization grant previously provided.
(4) If the client is successfully authenticated, the AS issues an access token for the granted resource.
    - Issued tokens have an expiration date (few hours or days).
    - Tokens are associated with client credentials.
(5) The client uses the token with the RS.
(6) The RS gives an access to the requested resource.

3.3 Authorization & access delegation framework

3.3.1 *Introduction.*

Our proposed solution is based on OAuth 2.0 and adapted for a cloud usage including FPGA devices. This solution aims to provide an authentication solution for 4 entities simultaneously (FPGA, client, TA and CP) and achieves perfect isolation between the client and the CP.

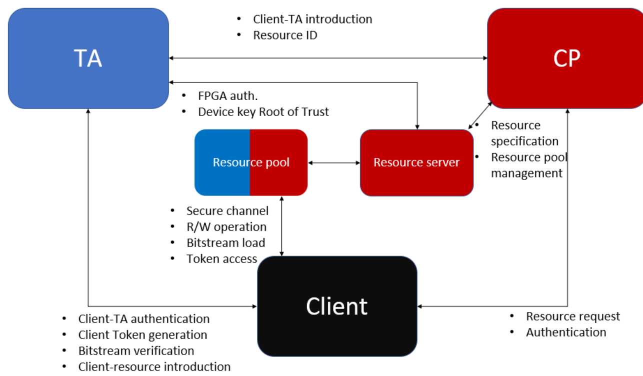

Figure 2: High level view of our solution.

The protocol seen in Section 3.2 must be adapted for the cloud use case for several reasons. In the first case, the RO was using the client's website and was willing to share information with the client who would then ask authorization for resource access and start the whole access delegation procedure. This scheme is no longer valid in an FPGA-accelerated cloud use case as the client uses RO's website to earn access to hardware owned by the RO.

In this situation, the cloud provider is considered as the resource owner, the trusted authority is referred as the authorization server and the FPGA is a part of the resource server.

Fig. 2 is a high level view of the proposed solution. Colors show the responsibility and the decision making of the respective entity. For example, the resource pool is split between the TA and the CP because the TA has an interface with each FPGA (e.g., PUF, shell).

3.3.2 *Client request & certificate creation.*

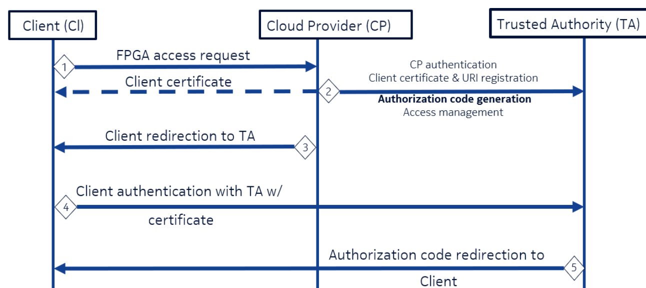

Figure 3: CP introduces the client to the TA, generates and manages authorization code. Client authenticates himself with the TA and obtains its authorization code.

To request resource from the CP, the client first sends a message to the CP's user-agent as shown in step 1 of Fig. 3. In this request message, the client includes its identifiers, certificate (or produces it online as stated below) and a redirection Unique Resource Identifier (URI). Those information are sent to the TA on the next step. The URI is used by the TA to send back redirected messages to the client via the CP's user-agent.

There are two different scenarios for client certification. Upon receiving the client's resource request, the CP would authenticate himself to the TA with its certificate, request a certificate for the client and share it with the client as shown on step 2 in Fig. 3. The client certificate is created from the CP's website. The client has to interact with the web browser to create the certificate and add randomness to the generated keys. A similar mechanism is seen in Microsoft Azure's key-pair generation for SSH channel protection.

It is also possible to create the client certificate offline from this protocol. The client would have the responsibility to generate a key-pair for himself. By doing that, the client would make the resource request to the CP with its certificate. Then, the certificate generation step would be skipped and as a result the protocol would be faster.

3.3.3 *HTTP redirection and authorization grants.*

Still on the second step in Fig. 3, after the CP's authentication, the client's request is accepted or declined. If the request is accepted, the TA generates the authorization code. By using the previously provided URI, the TA redirects the CP's user-agent back to the client to authenticate him directly. By doing that, the CP has authenticated and introduced the client to the TA.

At this time, the TA knows the client's certificate and the authorization code associated to the client's identifiers. To obtain its authorization code, the client needs to authenticate with the TA for the first time. A certificate-based TLS authentication is done [11]. If the client credentials are valid, the authentication is successful and the TA sends an HTTP redirection code to the CP's user-agent (HTTP code 302) alongside the client redirection URI. The client receives the authorization code from the CP's user-agent on the last step of Fig. 3.

In this protocol, the authorization code cannot be used as an attack vector. In fact, the authorization code is associated to a client credentials and URI. It is not a secret code because the CP's user-agent shares the authorization code through the HTTP redirection. The authorization code can be found in the user-agent history. In case of an authorization code redirection attack, which aims to get a backdoor access to the client's resource, a simple redirection URI check from the TA is sufficient. The URI used when requesting the authorization code must match the URI used for the access token generation as explained in Section 3.3.4. Hence, a malicious client cannot gain access to resources attributed to another client by intercepting the authorization code.

The role of this code is to ensure that the CP is authenticated and cannot be impersonated. By using the CP's user-agent to redirect the code, it can be confirmed that the CP which authenticated himself and gave authorizations for resource allocation to the TA is the same entity that communicates with the client in the protocol.

3.3.4 *Access management & token generation phase.*

After receiving the redirected authorization code, the client needs to authenticate himself again with the TA with its certificate, authorization code and redirection URI to request an access token from the TA. A certificate-based TLS authentication is done one last time to confirm client identity. This second authentication is necessary because it is a security measure to verify the client's identity.

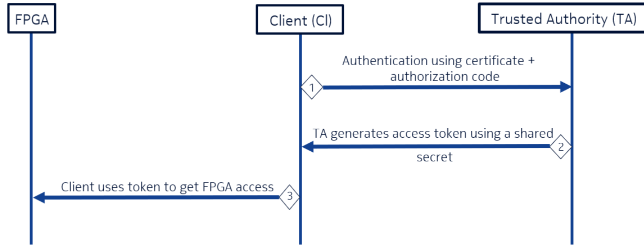

Figure 4: The TA generates the access token for the client.

The client needs to submit its authorization code to request the access token. If the client’s credentials are registered and associated with the authorization code used, the TA will be able to authenticate the FPGA device and generate the access token using a shared secret with the FPGA as shown in Fig. 4. Access tokens have scopes and duration of access. They are managed by the RO and endorsed by the TA [2]. These options are requested by the client during the first step of Fig. 3. Then, the RO accepts or declines the requested scopes and duration and notifies the TA (i.e., authorization server) on step 2. The client can decline an issued token if the scope requested does not match with its requests.

The access token’s content may also be extended to contain information like FPGA serial number, partially reconfigurable region identifier and so on. This feature gives flexibility for the implementation phase as additional mechanisms can be developed. Moreover, thanks to the shared secret and the TLS session between the client and the FPGA, a secure and tokenized confidential remote access can be set up for the client. The CP is isolated from the client’s computation but can still manage access scopes and duration.

3.4 Client & FPGA secure channel

After the token issuance, the client contacts the FPGA to earn access for resources he has been authorized. A TLS session is set up for secure communication with perfect forward secrecy between the FPGA and the client. The client and the FPGA create their shared secret with algorithms like DHE, ECDHE and then use symmetric encryption algorithms like AES-256-GCM. Once the TLS connection is established, the client sends its token to be authenticated. The FPGA proceeds to token parsing and gives access to the resources the client is authorized to. Further communications between the client and the FPGA will be encrypted. User privacy will be greatly enhanced and isolation from other entities will be achieved.

3.5 Access Control using OAuth 2.0 tokens

When all the entities are authenticated and the authorizations are granted, the client can access the FPGA with the access token. The token needs to be decrypted, parsed and actions are taken by the FPGA to program and allocate resources inside the device. Thanks to its resource server, the CP explicitly specifies the attributed resource information to the TA during the second step of Fig. 3.

According to OAuth 2.0 protocol, token content can be extended according to user preference [2]. to take advantage of this feature in a cloud FPGA context, critical information needs to be selected. These information will reinforce the access control of the client and ensure device/infrastructure security. It is up to the CP to decide the content of the token, but we think that the following details should be added to an issued token. Those information include FPGA serial numbers (or a specific challenge-response pair for a PUF) and some sort of partial reconfiguration region (PRR) identifier in case of a multi-tenant FPGA usage. A client identifier could also be useful to use accelerator blocks placed inside the FPGA. These are useful to identify the device, the client and the allocated PRR. Additionally, bitstream identifications, and bitstream signatures are stored in the token. The FPGA would be able to verify if one specific bitstream is cleared to be used for reconfiguration. The token must have validity timestamps for the FPGA to take action upon token expiration.

4 PERFORMANCE ANALYSIS

4.1 Theoretical Performance

TLS v1.2 algorithm implementations				
Related work	Approach	Paradigm	LUT	Total time
[12] Bellemou et. al	SW/HW	IoT/low cost	8503	67.5 ms
[13] Hamilton et. al	HW	High performance	90644	0.62 ms
[14] Wang et. al	HW	High performance	39052	11.3 ms

Table 1: Table shows different TLS 1.2 implementations with approaches and paradigms.

The theoretical hardware resource and latency used by a TLS implementation is greatly impacted by the implementation as seen in Table 1. The total time in Table 1 is for a single TLS handshake. For a commercial FPGA-enabled cloud usage, the hardware overhead should be minimal but the latency as well. The implementation of TLS should be optimized for cryptographic function accelerators and recurrently used functions and kept lightweight for other blocks like key exchange protocols. The purpose of Table 1 is to provide a theoretical basis and benchmark for future implementations of the novel authentication solution proposed in this paper. Analytically, we can show that the required time to generate the authorization code as described in Fig. 3 is as follows:

$$t_{code} = t_{Cl}(auth.) + t_{CP}(internal + auth.) + t_{TA}(internal) + t_{Cl-CP}(net.) + t_{Cl-TA}(net.) + t_{TA-CP}(net.) + (t_{cert}) \quad (1)$$

Internal tasks are not computationally expensive, for example, $t_{CP}(internal)$ refers to the time needed by the CP’s virtualization tools to check for available resources and allocate them to a client. The TA would only read/write values and generate the authorization code during $t_{TA}(internal)$. The certificate creation time t_{cert} can be skipped if the client already has a certificate. $t_{client}(auth.)$ and $t_{CP}(auth.)$ are respectively the client and the CP authentication as explained at the beginning of Section 3.3.3. $t_{net.}$ represents the cumulated network latencies and transport. Assuming that every entity has a low latency with each other (i.e., small $t_{A-B}(net.)$ where A and B are the communicating entities), the client could obtain the authorization code in few seconds. The time required for the token generation phase shown in Fig. 4 can be described as follows:

$$t_{token} = t_{Cl}(auth.) + t_{TA}(internal) + t_{Cl-TA}(net.) + t_{FPGA-TA}(net.) + t_{FPGA}(internal) \quad (2)$$

t_{token} should be very small because there is only a certificate-based TLS authentication of the client and an FPGA authentication using either a PUF or by reading a value stored in a secure FPGA memory. Then the TA proceeds with the token generation procedure, that involves few read and write operations. Every bit of information necessary for the token generation is stored in the TA's database. It is important to note that the value $t_{FPGA-TA}(net.) + t_{FPGA}(internal)$ is only for a PUF use case. If the TA has another authentication solution, these delays are no longer present in the token generation.

$$t_{access} = t_{client-FPGA}(net.) + t_{FPGA}(internal) << t_{code} + t_{token} \quad (3)$$

t_{access} is the time required to confirm that a generated token has a valid FPGA access. To confirm this, we need one TLS authentication between the client and the FPGA and a token verification which is represented by $t_{FPGA}(internal)$. This value is greatly smaller than the time required by our protocol to generate an authorization code and an access token. t_{access} requires less communications, whereas t_{code} and t_{token} include internal computing times and accumulated communication latencies between entities present in this protocol.

4.2 Time estimation for token generation

Let's set $t_{A-B}(net.)$ to 30ms and use the delay time of [12] in Table 1 for a standard TLS handshake (67.5 ms without network latencies). We have 3 Client-authenticated TLS handshakes in Eq. 1. There are 8 messages in a client authenticated TLS handshake which gives us $8x3x30 = 720ms$ for network latency and $3x67.5 = 202.5ms$ for TLS handshake. If the client generates its certificate offline and then requests an authorization code as stated in Section 3.3.2, $t_{code} = 720 + 202.5 = 922.5ms$. Tasks internal to CP and TA are not expensive, so they are much smaller against values taken into account for the previous estimation. We can round the previous value to 1000 ms for the worst case scenario with a relatively low cost TLS accelerator inside the FPGA.

Additionally for t_{token}, $2x8x30 + 2x67.5 = 682.5ms$ just for two TLS handshakes and network latencies without taking internal tasks into account. Most of the time is going to be spent by network latencies. During this phase we will have 3 message exchanges between the TA and FPGA for the PUF challenge-response pair and the token sent from the TA to the client. We have then $t_{token} = 682.5 + 3x30 = 772.5ms$.

As a final value we have $t_{code} + t_{token} = 1695.5ms$ to obtain authorization and generate an access token for the allocated FPGA. As a worst case scenario, this procedure should not last more than 2 seconds. Finally we can estimate with the following equation : $t_{access} = 30 + 67.5 + t_{FPGA}(internal) = 97.5ms + t_{FPGA}(internal)$. Accessing an FPGA with a valid token is approximately 100 ms. For this use case, Eq. 3 is true.

5 CONCLUSION

Static designs are necessary in FPGA-enabled clouds to implement the CP's FPGA shell. Other FPGA system primitives can also be needed to enable partial reconfiguration. The static design could be shared between the TA and the CP as proposed in [6]. By doing that, the TA could implement the logic necessary to build the OAuth 2.0-based solution presented in this paper. The TA would have a design close to a hardware security manager where token mechanisms would be implemented (token expiry, actions to take, managing keys, PUFs, cryptographic function accelerators...). This aspect needs further developments in future works.

Even if this paper focuses on authentication and resource allocation, the proposal is generic enough to be used in a multi-tenant context. Multi-tenancy in FPGA allows multiple users to share the same device at the same time. Each user is attributed a PRR, and they can only use their own area. Currently, multi-tenancy in FPGA is an active field of research and no commercial cloud provider uses this technology. Common multi-tenancy problems are resource sharing and user isolation. By combining PRR identifiers and token-based FPGA access, multi-tenancy can be enabled with our solution. The same thing can be said for services like bitstream certification provided by the TA.

This paper proposes a novel OAuth 2.0-based authentication and access delegation scheme for FPGA-enabled commercial cloud computing. Client's authentication protects its sensitive information and the CP's cloud infrastructure from malicious behaviors caused by identity theft. Furthermore, by introducing a trusted authority, the client's FPGA access and sensitive operations are isolated from the CP. The client benefits from a low latency single sign-on authentication for its FPGA thanks to a tokenized access. Security and privacy are enhanced for both the cloud provider and the client. For future works, the performance of the presented solution will be investigated.

REFERENCES

[1] F. Turan, S. S. Roy and I. Verbauwhede, "HEAWS: An Accelerator for Homomorphic Encryption on the Amazon AWS FPGA," in IEEE Transactions on Computers, vol. 69, no. 8, pp. 1185-1196, 1 Aug. 2020

[2] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749

[3] J. Vliegen, M. M. Rabbani, M. Conti and N. Mentens, "SACHa: Self-Attestation of Configurable Hardware," 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2019, pp. 746-751

[4] Fei Chen, Yi Shan, Yu Zhang, Yu Wang, Hubertus Franke, Xiaotao Chang, and Kun Wang. 2014. Enabling FPGAs in the cloud. In Proceedings of the Computing Frontiers Conference (CF'14)

[5] Mikhail Asiatici, Nithin George, Kizheppatt Vipin, Suhaib A. Fahmy, and Paolo Ienne. 2017. Virtualized execution runtime for FPGA accelerators in the cloud. IEEE Access 5 (2017), 1900–1910

[6] H. Englund and N. Lindskog, "Secure acceleration on cloud-based FPGAs – FPGA enclaves," 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), New Orleans, LA, USA, 2020, pp. 119-122

[7] M. E. S. Elrabaa, M. Al-Asli and M. Abu-Amara, "Secure Computing Enclaves Using FPGAs," in IEEE Transactions on Dependable and Secure Computing, vol. 18, no. 2, pp. 593-604, 1 March-April 2021

[8] Oliver Knodel, Paul R. Genssler, Fredo Erxleben, and Rainer G. Spallek. 2018. FPGAs and the cloud–An endless tale of virtualization, elasticity and efficiency. Int. J. Adv. Syst. Meas. 11, 3 (2018).

[9] Ken Eguro and Ramarathnam Venkatesan. 2012. FPGAs for trusted cloud computing. In Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL'12). 63–70

[10] B. Hong, H. Kim, M. Kim, T. Suh, L. Xu and W. Shi, "FASTEN: An FPGA-Based Secure System for Big Data Processing," in IEEE Design & Test, vol. 35 Feb. 2018

[11] Rescorla "The Transport Layer Security Protocol Version 1.3", RFC 8446, 2018

[12] A. M. Bellemou, A. García, E. Castillo, N. Benblidia, M. Anane, J. A. Álvarez-Bermejo, and L. Parrilla, "Efficient Implementation on Low-Cost SoC-FPGAs of TLSv1.2 Protocol with ECC AES Support for Secure IoT Coordinators," Electronics, vol. 8, no. 11, p. 1238, Oct. 2019

[13] Hamilton, M.; Marnane, W.P. Implementation of a secure TLS coprocessor on an FPGA. Microprocess. Microsyst. 2016, 40, 167–180.

[14] Wang, H.; Bai, G.; Chen, H. A Gbps IPSec SSL Security Processor Design and Implementation in an FPGA Prototyping Platform. J. Signal Process Syst. 2010