Decentralized Policy Enforcement in Zero Trust Architectures

Lars Creutz
Institute for Software Systems
Trier University of Applied Sciences
Birkenfeld, Germany
Email: l.creutz@umwelt-campus.de

Guido Dartmann
Institute for Software Systems
Trier University of Applied Sciences
Birkenfeld, Germany
Email: g.dartmann@umwelt-campus.de

***Abstract*—The introduction of the Zero Trust Architecture (ZTA) provides continuous improvement in the security of modern networks, replacing older approaches such as perimeter-based networks that rely on firewalls or VPN connections. In our paper, we describe how the basic design of a ZTA can nevertheless lead to security problems, since decisions are usually made at a central point in the network. We present an approach on how this decision can be made in a decentralized manner by the components of the network itself, in order to be able to increase the security of the component and the security of the ZTA. We demonstrate our approach in an Edge-Cloud scenario and describe all components of the ZTA, for which we provide an open source implementation.**

I. Introduction

In the rapidly evolving digital landscape, cybersecurity has emerged as a critical concern for organizations of all sizes and across all industries. As systems grow increasingly complex, the traditional perimeter-based security model, where internal resources are trusted and external ones are viewed with suspicion, has proven inadequate. Zero Trust Architectures (ZTA) [1] is therefore seen as a new approach to a security model.

The ZTA is a security model that operates on the assumption that no user or device, whether inside or outside the network perimeter, should be automatically trusted. Instead, every access request must be validated and authorized, based on a comprehensive understanding of the user, device, application, and data. This approach reduces the potential attack surface and makes it harder for adversaries to move laterally within the network.

In general, ZTAs can be broken down into three main components: assets, the Policy Enforcement Point (PEP), and resources [2]. The communication in ZTA is divided into two different zones: untrusted up to the PEP and implicitly trusted from the PEP [2]. The PEP in the view of the National Institute of Standards and Technology (NIST) [2] is a central point in the architecture and therefore an attack point that has often received little attention in the past. A centralized PEP creates a single point of failure, where if the PEP is compromised, an attacker can potentially gain unrestricted access to network resources. Additionally, in distributed and cloud-based environments, centralized enforcement can introduce latency and inefficiencies due to the need to route all traffic through a single point for inspection and policy enforcement. This is particularly problematic in the case of DDoS attacks, which can disrupt the entire ZTA in the event of a successful attack on the PEP. Implicit trust zones one the other hand are areas within a network where all entities are automatically trusted once they have been authenticated and granted access. This, again, could allow for lateral movement inside the network and exposes resources to various kinds of threats.

While centralized policy enforcement and implicit trust zones have been standard practice in traditional network security, they can create vulnerabilities and inefficiencies when applied within a Zero Trust model.
Our approach instead advocates for decentralized enforcement and a lack of implicit trust, making it a more resilient and effective model for today's complex, distributed networks, especially when applied to areas where primarily managed devices are used. We address the mentioned and other problems of centralized ZTA and describe why decentralized approaches can be useful. In addition, we describe how to use free/low-cost yet highly available infrastructure from well-known git providers to manage policy in a secure and decentralized manner. Furthermore, we use cyber-physical contracts within our ZTA to describe processes in an understandable way and to securely exchange data. We begin by describing related works and the contributions of the paper. Section II then explains our decentralized policy enforcement approach and how the stakeholders of a traditional ZTA operate in our context. In order to illustrate and test our approach, we describe an implementation of an example service with decentralized policy enforcement in Section III. Section IV summarizes the work and provides an outlook for further improvement and application of our decentralized policy enforcement architecture.

A. Related Works

A well-known approach to ZTA is given by NIST in [2]. Their fundamental description of ZTA we later adopt as state of the art and compare our approach to improve ZTA in general through decentralized enforcing of policy. Another well-known approach is BeyondCorp [3], the Zero Trust model of Google. The authors describe the aforementioned problem of enforcing perimeter security and challenge the use firewalls to secure a network. To prevent lateral movement within a network, Google here waives special privileges on the intranet and continuously verifies each stakeholder per resource. BeyondCorp is primarily considering managed devices that join unprivileged networks via a certificate and RADIUS[1] while unmanaged devices are assigned in a guest network. Communication flows through an ”access control engine” that manages access to resources behind it, similar to the PEP.

There are some works that contain decentralized approaches to policies across different areas. The authors in [4] describe a Decentralized Policy Information Point model to enforce policies across different domains with centralized PEPs. Among other things, [5] describes how policies can be distributed in order to check multiple policies when accessing a single resource. However, the work does not refer specifically to ZTAs.

To the best of our knowledge, there are no approaches to decentralized ZTA with policy enforcement at the resource level in the literature. However, some works criticize components of centralized ZTA that are improved by our work through decentralization. The authors in [6] describe challenges and security risks, an excerpt of which is presented below, which are improved by our ZTA:

a) PEP as new attack surface: The authors name attacks like DDoS, route hijacking, or supply chain attacks on the PEP as a core part of the ZTA. Centralizing the PEP can make for major system failures if all requests have to be routed through this component. Furthermore, the PEP can be a popular target for advanced attacks to then route false requests to the resources. Our approach reduces these attack opportunities by having resources handle the function of the PEP and managing the policy in a decentralized manner.

b) Unauthorized policy changes: The authors in [6] share concerns from [2], where unauthorized policy changes can cause damage in a ZTA. Remedies are described in constant monitoring of policy changes and auditing of the enforcement code [2]. We use several cryptographic methods in our ZTA to safeguard and track policy changes.

c) Vendor lock-in: The authors in [6] criticize possible vendor lock-ins and describe IoT scenarios and incompatibilities between operators and the problem of transferring this to ZTA components. We use well-established standard software in our ZTA and different providers in parallel from the beginning, which makes an extension, reduction or change seamlessly possible.

d) Standardization: Here, the authors criticize that there is little standardization in the area of policy management [6]. Various approaches exist in the literature, such as in [7], where a policy language is defined to specify firewall rules. The work builds on several other approaches to describe a access control, for example on a markup language [8] and a risk-adapable model [9]. In addition, there are various Software as a Service offerings from well-known manufacturers that rely on their own proprietary implementations, such as Cloudflare [10] and Google Cloud [11]. This further reinforces the aforementioned point of vendor lock-in. Our approach uses json as format and is therefore widely applicable and human readable.

[1]https://datatracker.ietf.org/doc/html/rfc2865 (Accessed October 2023)

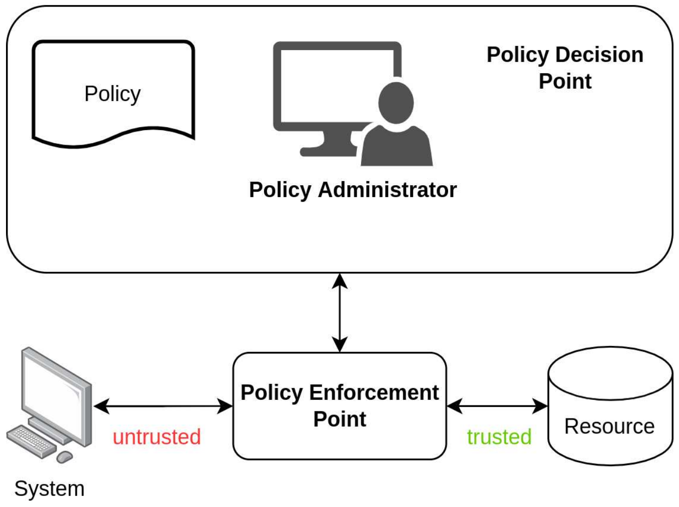

Fig. 1. General structure of a ZTA referring to NIST [2].

B. Contribution

Below we briefly outline the contributions of the paper:

a) Patti: With the paper we also publish the *Patti* [12] project open source under the MIT license. *Patti* allows the creation and processing of *Cypher Social Contracts* [13] without the direct use of *Fides* [14]. We use cyber-physical contracts for authentication which allows to describe the processes in a human-readable way and creates understandability when observing the actions of stakeholders inside the network.

b) (Edge-)Device Authentication using Cypher Social Contracts: We explain the approach of authenticating devices to resources with secure *Cypher Social Contracts* [13]. For this purpose, we use *Patti* to create contracts between assets and resources using decentralized policy information.

c) Decentralized Policy Model for Zero Trust Applications: We introduce a method for describing and using a Zero Trust Policy in a simple, cost-effective, and secure manner. We show how individual resources and assets use the policy and explain advantages compared to the traditional approach of ZTAs.

d) Decentralized Policy Enforcement on Resources: We further show how policy enforcement within a Zero Trust Architecture can be achieved by the resources themselves. In this context, we explain how our approach differs from common ZTAs.

e) Policy Administration: We describe a secure approach to managing the decentralized policy. Different cryptographic methods are used for this purpose in order to prevent possible attacks on the policy administration.

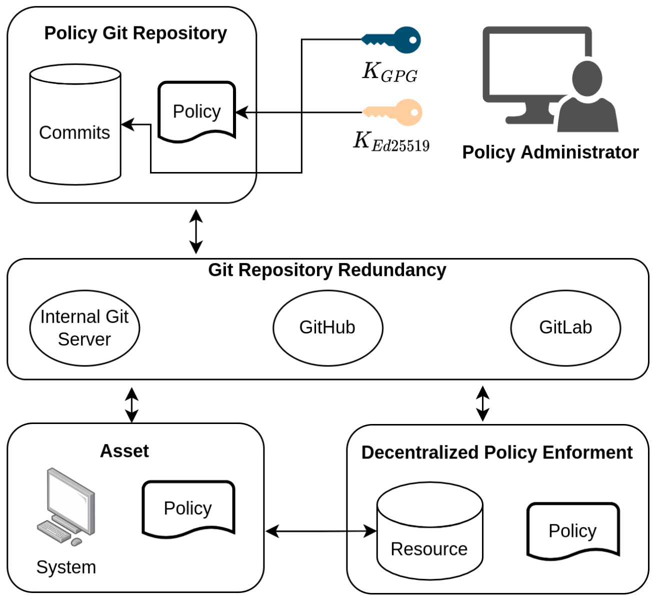

Fig. 2. Our approach for a decentralized ZTA where the resources themselves enforce the rules of the policy. The policy is secured by two signature methods and managed via redundant git operators.

II. Decentralized Policy Enforcement

A. Differences to NIST Zero Trust Model

Our approach differs from the NIST approach [2] and is characterized by more decentralization and enforcement of the policy at the resource level. We consider it problematic that the Policy Enforcement Point (PEP) must be trusted from a resource perspective, thus making the division into untrusted and implicit-trusted zone (see Fig. 1). The approach is useful when Zero Trust principles are applied to organizations with a high density of legacy services. Here, security is generally improved without generating an immense workload and adjusting all services. For a new development of a zero trust architecture, however, it should be considered whether the PEP can be designed in a more decentralized way, as in our approach (see Fig. 2). The security of resources is improved by the elimination of a central PEP, since no component per se needs to be trusted. This also strengthens the quality of the resource's implementation and prevents careless errors in its management.

Furthermore, we concretize an approach how publicly available infrastructure like GitHub and GitLab can be used to describe, distribute and decentralize the policy in a ZTA. Using git in general prevents the vendor lock-in mentioned in [6]. The methods we suggest for the respective providers are generally offered by all git operators, for example rules that commits must be signed and must originate from a certain party. Furthermore, our approach uses redundant providers right from the start, which makes it easier to switch. If a git provider can no longer be used, it can be easily replaced without the system suffering any downtime.

Other attack vectors described in [6] are DDoS, route hijacking or supply chain attacks. Since in our approach the resources themselves are responsible for enforcing the policy and thus a central PEP is not required, the system as a whole is more difficult to attack via DDoS. Furthermore, the resources can be scaled individually, which is only possible to a limited extent with a central PEP, since all requests must first be forwarded via it. Our approach therefore makes it easier to scale dynamic services more easily and securely, which can also lead to a reduction in cloud costs.

B. Policy Definition

The entire policy is defined in a single json file that is used by every stakeholder of the architecture. Utilizing json as a human readable format allows for easy administration of the policy. To give a brief example, the policy for our example scenario describing the Edge-Cloud device and the used resource is defined in Listing 1.

```
{
  "devices": [
    {
    "device-id": "ozy-1",
    "services": [
      {
        "service-id": "ventilation",
        "post": "upload,auth",
        "get": "current_model,policy"
      }
    ],
    "public_key": "PVQaczM7DeVFRxnnVU+
        ↪ gLIoUjLU4BIbtaWByWqupYBg=",
    "room": "9925-115"
  } ],
  "services": [
    {
    "service-id": "ventilation",
    "public_key": "yAPHMzJP3cyNSk6D+
        ↪ uaMqrNWQyLmbnH/3clBZsGfoNE=",
    "template": "a8cc[...]ba6f",
    "ip": "localhost",
    "port": "5000"
    }
  ]
}
```

Listing 1. Policy definition in our example scenario.

Defining the policy in this way offers several advantages: The configuration effort of assets can be reduced by keeping the entire architecture of the system in one central point. In addition, services can be easily changed if, for example, they undergo a change of their internal IP address. Internal DNS servers can even be omitted, which minimizes further attack vectors like DDoS or cache poisoning [15] because assets directly use the IP:Port combination to reach the resources.

This is particularly useful and suitable for our scenario, as the Edge-Cloud devices are centrally managed and thus no unmanaged devices are part of the system. We describe the services using simple REST APIs and specify for each asset which route (GET, POST) it is allowed to call on that service. Other approaches such as Remote Procedure Calls could be implemented accordingly. The definition of a service also contains a public key and a hash of a *Template* from which a *Cypher Social Contract* [13] can be created for authentication to the respective resource. *Templates* define the processes and responsibilities within a *Cypher Social Contract* [13]. Accordingly, different approaches can be used for authentication. We demonstrate this using an authorization token based on a minimal *Template*. Other mechanisms, such as multi-factor authentication, can be incorporated into the *Template* in a similar way.

C. Policy Administration

In order to configure the policy, the policy administrator modifies the policy file described above. Every change must be secured by two signatures. First, the hash of the new policy file is signed ($K_{Ed25519}$) [16], followed by the git commit that synchronizes the policy changes (K_{GPG}). The changes are then synchronized with the git repositories. Here, the git operators check whether the respective commit has a correct signature. This is a first hurdle for possible attacks. Even if an incorrectly signed commit reaches an asset or resource through the manipulation of a git operator, the signature is validated again locally and it is ensured that every commit of the repository is signed by the known GPG key. Should an attacker control the GPG key, the second signature method (Ed25519) provides further security. Only when both signatures have been validated the policy update is considered valid. Accordingly, the two signature keys should be managed by two different systems/persons. Our approach therefore provides solutions to the problems mentioned before for unauthorized policy changes.

D. Assets and Authentication

Assets represend the Edge-Cloud device in our Zero Trust Architecture and communicate with a service (considered a resource). The devices are managed devices that are easy to configure:

- Import the corresponding public key to K_{GPG}.
- Create an account for the usage of *Cypher Social Contracts* [13] on device. An account consists of a private (secret) key and a public key and, like the key $K_{Ed25519}$ of the policy administrator, operate on Curve25519 [17].
- Include the policy in the local application code. This can be boilerplate code in a shared library within any organization.
- Use the policy information for the application context: Which endpoint to contact, which *Template* to use to create a *Cypher Social Contract* [13].

The enumeration shows that the configuration of the individual devices can be automated quite easily and it then only depends on the use case in various contexts where application-specific code must be implemented. After a device has been configured correctly, the policy administrator just needs to add it to the policy, describe its specific permissions per resource, and publish an update to the policy.

E. Resources and Policy Enforcement

Resources correspond to services within a network that are used by users or assets. In our ZTA, we assume that resources are accessed via REST APIs. One of the responsibilities of a resource is to ensure that the current valid policy is being used, which means that each resource within our ZTA periodically checks to see if an update has been pushed out by the policy administrator. This process can be simplified by implementing a special REST (POST) route per resource, which allows to react on push events of the used git providers. Here webhooks are used (see e.g. GitHub [18] or GitLab [19]) to tell the resource that the repository has been updated. To update their local repository with the policy, resources perform the following operations:

- Perform *pull* operation on local repository. If the process exits with a nonzero exit code, indicating an error, do not update the policy. This ignores potentially dangerous changes, since even a forced *push* operation by the policy administrator will result in an error. Accordingly, the history of the policy is clearly trackable and every change is verifiable.
- For each commit (new and previously known) verify the signature using the *verify-commit*[2] command. This prevents multiple commits from being added by the policy administrator, but only the last one having a correct signature.
- Verify the second signature (issued by $K_{Ed25519}$) of the update policy file. The check is performed with a custom bash script that utilizes the Fides [14] command line interface to check the signature against the SHA256 hash of the policy file.

After the policy update, the resources each update the permissions of the assets for the resource itself. Again, updating the policy can be easily shared across organizations in the form of a shared library, making it easy to integrate and maintain. In order to check a specific endpoint within the API against the policy, the resource uses the decorator *apikey_required* which checks per route whether the passed token is valid for the respective asset. The decorator thus passes an overview of the asset to each function call per endpoint that needs to be protected. This approach allows for holistically secure resources on the network that do not rely solely on the assumption of a secure PEP, but must themselves be securely implemented.

III. Evaluation

A. Scenario

Our scenario describes the establishment of a novel ZTA in the area of Edge-Cloud computing. For this we use a specially

[2]https://git-scm.com/docs/git-verify-commit (Last access June 2023)

designed Edge-AI board that combines a microcontroller with a Raspberry Pi. The Raspberry Pi takes over more computationally intensive tasks here and supports the microcontroller. In our experiments we have simulated this setup. In general, the board is currently under continuous development and a paper will be published in the near future. The selected service (cloud) should monitor ventilation events and will be used in the future to combine local machine learning (Edge) with distributed learning (via the cloud through different rooms). The tasks of the individual components will now be briefly summarized:

Raspberry Pi:

- Authenticate in the Zero Trust Architecture.
- Train local (Edge) ML models for microcontroller.
- Flash microcontroller with newest software.
- Perform model updates from the cloud if necessary (if different rooms have better ventilation models).
- Enter low-power state and instruct microcontroller to take over operation.

Microcontroller:

- Run ML model and recognize ventilation events.
- Send ventilation data to cloud service.
- Wake Raspberry Pi if necessary (model update, re-authentication required)

B. Implementation

The source code is freely available in [20] and [12]. The simulation contains the following components which are explained in more detail below:

- **Patti**: Repository of the *Patti* library to create *Cypher Social Contracts* [13] without running *Fides* [14] .
- **OZY-Sim**: Simulator that mimics our custom Edge-AI board.
- **Ventilation**: Example services that acts as a resource in our ZTA.
- **Policy**: Policy repository that also contains the policy administration scripts.

C. Authentication with Cypher Social Contracts

The entire authentication process is shown in Fig. 3 and is performed by the Raspberry Pi. In accordance the following steps are carried out:

- Create a *Cypher Social Contract* [13] using the *Template* of the resource. Here, *Patti* is used to create and sign a transaction, which is then sent to the resource via the *auth* API endpoint.
- Since our device is allowed to access the resource according to the policy, the resource responds with two transactions: the acceptance of the contract and the confirmation of the contract task, which contains the encrypted authentication token, which is later used in the API calls of the microcontroller. The two transactions are applied to the local contract using *Patti*, which checks the signature of the resource.

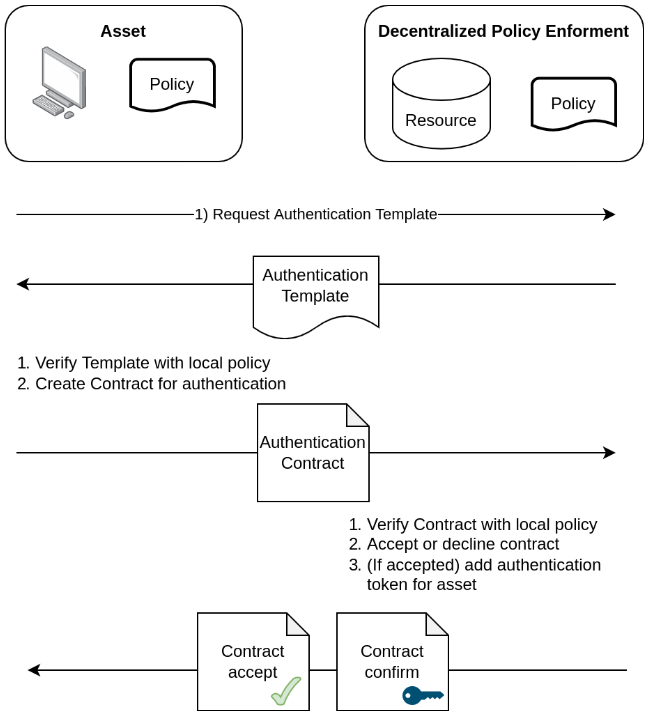
Asset
Policy
Decentralized Policy Enforment
Resource
Policy
1) Request Authentication Template
Authentication Template
1. Verify Template with local policy
2. Create Contract for authentication
Authentication Contract
1. Verify Contract with local policy
2. Accept or decline contract
3. (If accepted) add authentication token for asset
Contract accept
Contract confirm

Fig. 3. Process of authentication to a resource. Assets obtain the hash of the *Template* via the global policy to create a contract for authentication. If the resource itself successfully verifies that the asset has access to the resource, the resource accepts the contract and sends an updated token to the asset.

If authentication is successful, the Raspberry Pi transmits the authentication token to the microcontroller and enters a low-power state.

D. Regular operation

After the Raspberry Pi has switched to the low-power state, the microcontroller takes over the regular operation of the device and sends the measured events to the resource. Within our simulation we use placeholders, which are transmitted at regular intervals via the endpoint *upload*. Each call to a resource is secured by HTTPS and contains the following HTTP headers:

- *X-AUTH-TOKEN*: The authentication token, which was communicated to the Raspberry Pi via the *Cypher Social Contract* [13].
- *X-DEVICE*: The ID of the asset according to the policy (here ozy-1).

For testing purposes, our resource only issues authentication tokens that have a validity of 10 seconds. Accordingly, the microcontroller checks the response of the resource to the transmission of the event. If the resource tells the device that it needs to re-authenticate, the Raspberry Pi is switched on and instructed to create a new *Cypher Social Contract* [13].

E. Policy updates

In order to validate our system locally and not necessarily embed it with various git operators, there is an endpoint for testing a policy update, which is called locally. When such a POST request is made to the endpoint *policy_update*, the previously described steps for updating the policy from a resource perspective are performed. Furthermore, the sample policy repository contains a script for administration. After the policy file is modified locally, the following operations are initiated on behalf of the policy administrator:

- Load $K_{Ed25519}$.
- Calculate the SHA256 hash of the policy file.
- Create the signature in the format that can be validated using the Fides [14] command line interface.
- Add the changes to the repository of the policy.
- Commit the changes and sign the commit using K_{GPG}.

If the system is to be migrated to a production scenario, the route used, which is then called by the git providers via webhooks, should also be secured by an API key. This key can be stored accordingly with the git providers so that other parties cannot call the route and only valid requests (after the policy has been updated) trigger an update of the local policy.

IV. Conclusion and future work

In summary, our paper presented a novel approach to ZTA that combines secure cyber-physical contracts, decentralized enforcement of a Zero Trust policy, and the use of highly reliable git operators. Initially, we explain our approach in conceptual terms and how it differs from the state of the art in research. We also outline a simulation, which we have made open source, that has translated our approach into practice. In the future, we plan to further refine the approach and apply it to the Edge-AI board to implement a real-world use case in an Edge-Cloud continuum.

Acknowledgment

This work has been funded by the Federal Ministry for Economic Affairs and Climate Action (BMWK) under the grant number 01MD22002D.

References

[1] J. Kindervag, "Build Security Into Your Network's DNA: The Zero Trust Network Architecture," Forrester Research.

[2] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero trust architecture," 2020-08-10 04:08:00 2020. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=930420

[3] R. Ward and B. Beyer, "Beyondcorp: A new approach to enterprise security," *;login:*, vol. Vol. 39, No. 6, pp. 6–11, 2014.

[4] M. R. Rahman, A. S. Shahraki, and C. Rudolph, "Decentralized policy information points for multi-domain environments," 2021.

[5] M. Cavage, Y. Xiao, and B. J. Behm, "Distributed policy enforcement with optimizing policy transformations," 12 2010, US Patent US9237155B1. [Online]. Available: https://patents.google.com/patent/US9237155B1/en

[6] S. Teerakanok, T. Uehara, and A. Inomata, "Migrating to zero trust architecture: Reviews and challenges," *Security and Communication Networks*, vol. 2021, p. 9947347, May 2021. [Online]. Available: https://doi.org/10.1155/2021/9947347

[7] R. Vanickis, P. Jacob, S. Dehghanzadeh, and B. Lee, "Access control policy enforcement for zero-trust-networking," in *2018 29th Irish Signals and Systems Conference (ISSC)*, 2018, pp. 1–6.

[8] OASIS, "eXtensible Access Control Markup Language (XACML) Version 3.0," http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf (Accessed July 2023).

[9] R. McGraw, "Risk-Adaptable Access Control (RAdAC)," https://csrc.nist.gov/csrc/media/events/privilege-management-workshop/documents/radac-paper0001.pdf (Accessed July 2023).

[10] Cloudflare, "Policies," https://developers.cloudflare.com/cloudflare-one/policies/ (Accessed July 2023).

[11] Google Cloud, "Identity and Access Management documentation," https://cloud.google.com/iam/docs/?hl=en (Accessed July 2023).

[12] Lars Creutz, "Lars Creutz / Patti · GitLab," https://gitlab.rlp.net/l.creutz/patti.

[13] L. Creutz and G. Dartmann, "Cypher social contracts a novel protocol specification for cyber physical smart contracts," in *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, 2020, pp. 440–447.

[14] L. Creutz, J. Schneider, and G. Dartmann, "Fides: Distributed cyber-physical contracts," in *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, 2021, pp. 51–60.

[15] S. Ariyapperuma and C. J. Mitchell, "Security vulnerabilities in dns and dnssec," in *The Second International Conference on Availability, Reliability and Security (ARES'07)*. IEEE, 2007, pp. 335–342.

[16] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, Sep 2012. [Online]. Available: https://doi.org/10.1007/s13389-012-0027-1

[17] D. J. Bernstein, "Curve25519: New diffie-hellman speed records," in *Public Key Cryptography - PKC 2006*, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 207–228.

[18] GitHub, Inc., "Webhook events and payloads," https://docs.github.com/en/webhooks-and-events/webhooks/webhook-events-and-payloads (Accessed June 2023).

[19] GitLab, "Webhooks," https://docs.gitlab.com/ee/user/project/integrations/webhooks.html (Accessed June 2023).

[20] Lars Creutz, "Lars Creutz / Decentralized Zero Trust Policy Enforcement Paper · GitLab," https://gitlab.rlp.net/l.creutz/paper-dztpe.