

SA²FE: A Secure, Anonymous, Auditable, and Fair Edge Computing Service Offloading Framework

Xiaojian Wang, *Graduate Student Member, IEEE*, Huayue Gu, *Graduate Student Member, IEEE*,
Zhouyu Li, *Graduate Student Member, IEEE*, Fangtong Zhou, *Graduate Student Member, IEEE*,
Ruozhou Yu, *Senior Member, IEEE*, Dejun Yang, *Senior Member, IEEE*, and Guoliang Xue, *Fellow, IEEE*

Abstract—The inclusion of pervasive computing devices in a democratized edge computing ecosystem can significantly expand the capability and coverage of near-end computing for large-scale applications. However, offloading user tasks to heterogeneous and decentralized edge devices comes with the dual risk of both endangered user data security and privacy due to the curious base station or malicious edge servers, and unfair offloading and malicious attacks targeting edge servers from other edge servers and/or users. Existing solutions to edge access control and offloading either rely on “always-on” cloud servers with reduced edge benefits or fail to protect sensitive user service information. To address these challenges, this paper presents SA²FE, a novel framework for edge access control, offloading and accounting. We design a rerandomizable puzzle primitive and a corresponding scheme to protect sensitive service information from eavesdroppers and ensure fair offloading decisions, while a blind token-based scheme safeguards user privacy, prevents double spending, and ensures usage accountability. The security of SA²FE is proved under the Universal Composability framework, and its performance and scalability are demonstrated with implementation on commodity mobile devices and edge servers.

Index Terms—Edge computing, service offloading, security, anonymity, auditability, fairness

I. INTRODUCTION

As real-time computation-intensive applications such as metaverse [1], cloud gaming [2], and autonomous driving [3] continue to grow, service providers are increasingly deploying services closer to the users [4]. Edge computing can greatly improve user experience and reduce the cost of service providers, by achieving low latency, high reliability and backhaul communication efficiency. An increasing number of companies are entering the arena [5].

Meanwhile, the rise of the Pervasive Edge Computing (PEC) paradigm [6], which utilizes the computing capabilities of varied and decentralized devices as edge servers, accelerates the expansion of the edge server provider landscape, by democratizing the edge computing ecosystem and leveraging power of the crowd. A PEC ecosystem may involve many

large and small edge providers, including but not limited to telecom companies, road-side unit operators, private infrastructure owners, and even ad hoc providers such as individuals with spare computing devices [7]. In most cases, a telecom company provides a connection access point for edge providers and users, and usually an accompanied edge service discovery procedure for users to access the available services [8].

However, with the expansion of the edge computing ecosystem, and especially PEC with decentralized providers, both edge server owners and users encounter challenges in providing and utilizing trustworthy edge computing services. To foster the sustainable development of the edge computing market, it is imperative to design and develop technical approaches that can safeguard user rights, protect stakeholder interests, and maintain healthy competition.

In the PEC environment, a very important and indispensable part is service discovery, which is used to find available services nearby. In traditional service discovery within Named Data Networking or Information-Centric Networking, the requesting service identities are generally exposed to surrounding devices to efficiently allocate available services to the requesting users [9]. Service identities, such as names or types, if descriptive or inferable, could reveal information about the nature of the data or services, potentially exposing sensitive or proprietary information to anyone who can intercept this information [10]. For instance, the type of service requested by a user (such as service name or identifier) could be misused in various ways, such as profiling and identifying a user [11] or inferring sensitive user attributes (e.g., inferring that a user requesting video-based visual assistance has a visual impairment [12]). If intercepted or accessed by malicious entities, sensitive service names could be used to perform targeted attacks, including data breaches and service disruptions. Protecting these service types/names from unauthorized disclosure is critical to maintaining the integrity and reliability of the network. Some privacy-preserving service discovery approaches have been proposed to protect service request privacy [13], but only within traditional discovery environments. However, in a PEC environment, users also have this need but lack available solutions to protect service information. Additionally, users may not want to leak their identity, preferring to remain anonymous to protect their privacy.

In addition, given the diverse stakeholders involved in service offloading, fairness in the service offloading process is paramount. Fairness here refers to the equitable treatment of service requests among edge devices with comparable capabilities and minimal latency differences from the user’s perspective, made possible by service provider profiling and

Wang, Gu, Li, Zhou, Yu ({xwang244, ryu5, hgu5, zli85, fzhou}@ncsu.edu) are with North Carolina State University, Raleigh, NC 27606, USA. Yang (djiang@mines.edu) is with Colorado School of Mines, Golden, CO 80401, USA. Xue (xue@asu.edu) is with Arizona State University, Tempe, AZ 85287, USA. The research of Wang, Yu, Gu, Li, Zhou was supported in part by NSF grants 2045539, 2414523 and 2433966. The research of Xue was sponsored in part by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-23-2-0225. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

This work has been submitted to the IEEE for possible publication. Copyright may be transferred without notice, after which this version may no longer be accessible.

testing to ensure that multiple edge servers with similar abilities compete for task assignments. The assurance of fairness should not be determined by any single entity, be it the end-user, the base station overseeing offloading, or specific edge servers. Furthermore, ensuring financial accountability of the users, and appropriate compensation of the providers, is critical to ensuring longevity of the market. Unfortunately, the above security and privacy guarantees are lacking in existing works [8], [14], and offloading fairness and auditability have not been addressed in an ecosystem with untrusted parties.

In this paper, we design SA²FE, an innovative framework for anonymous, auditable, and fair service offloading in a PEC environment, addressing key challenges in service offloading posed by the presence of multiple competing edge server infrastructure providers. At the core of our approach is a rerandomizable puzzle primitive, which we define and design as the foundation of our framework to enable fairness in the offloading process. Building on this primitive, we propose a comprehensive framework that specifies detailed interaction protocols among all participating parties, ensuring offloading fairness while protecting service type privacy. To safeguard user identity while maintaining authorized access and accountability, we propose an anonymous token-based service request scheme. We design a rerandomizable puzzle-based scheme that allows offloading a service request to an eligible edge server without revealing any service-specific details to the offloading broker, or edge server-specific details to the user. We rigorously prove SA²FE's security and demonstrate its practicality on commodity devices.

Our contributions are summarized as follows:

- We propose SA²FE, a secure and efficient offloading framework that preserves the privacy of user identity and requested service type, ensures fairness in edge server selection, and incorporates auditing for accountability.
- We present a novel puzzle-based offloading protocol to protect service type confidentiality while ensuring fair and randomized edge server selection. Two implementations based on bilinear map and universal re-encryption respectively are proposed to realize the puzzle scheme.
- We propose a token-based service access scheme that maintains user and service type confidentiality while enabling accountable token verification and claiming.
- We formally prove the security of SA²FE under the Universal Composability (UC) framework.
- We implement and evaluate a prototype of SA²FE on commodity mobile and edge devices. The experimental results show that SA²FE has low computation and communication overhead and is efficient and scalable.

Organization. Section II reviews related work. Section III introduces the system models. Section IV gives an overview of SA²FE. Section V presents the detailed design of SA²FE. Section VI presents security analysis of SA²FE. Section VII shows its performance. Section VIII concludes this paper.

II. RELATED WORK

One related aspect of cloud and edge computing security is the access control problem. In cloud computing, access control mainly focuses on protecting security and confidentiality of

user data hosted on third-party cloud storage [15]. Some have studied secure and privacy-preserving data sharing through a centralized cloud [16], [17]. The cloud provider plays a central role in facilitating access control as the single party involved. APECS [8] is the first distributed, multi-authority access control scheme in a dynamic pervasive edge computing ecosystem. However, APECS mainly focuses on user data access control, neglecting anonymity and privacy preservation during service offloading, and fairness considerations. AADec [14] focuses on access control, prioritizing data exchange over offloading at the base station, with auditing confined to user data rather than service offloading.

User data privacy has been studied in either cloud or edge offloading. Li *et al.* [18] proposed a system for solving over-determined linear equations, ensuring privacy via permutation and validity through a detection algorithm. Mao *et al.* [19] proposed combining differential privacy and secure model weight aggregation to ensure privacy-preserving offloading of DNN training tasks. Chen *et al.* [20] proposed a secure outsourcing algorithm for modular exponentiations in the one-malicious version of the two untrusted program model.

Many have studied edge offloading focusing on improving offloading performance subject to limited resources, such as resource provisioning [21], task partitioning [22], task selection [23], load balancing [24], etc. Some recent works focus on task offloading in various edge computing scenarios using different methods, such as at intersections with game theory [25], in satellite networks using queuing theory [26] or game theory [27], in online offloading scenarios employing Deep Reinforcement Learning [28], in Aerial Mobile Edge Computing Networks through joint optimization [29], and using pairing theory to match services [30]. These methods assume trust among all parties and overlook privacy concerns.

To summarize, while existing work has addressed certain individual security concerns such as authentication, access control, user data privacy and location privacy, there lacks a comprehensive framework for service offloading in a democratized edge computing ecosystem that ensures offloading security, user identity and service anonymity, token accountability, and offloading fairness all at once. Our proposed framework SA²FE not only fills this gap and ensures secure offloading, but is also highly efficient, scalable, and compatible with commodity mobile devices and edge servers.

III. MODELS AND PROBLEM STATEMENT

A. System Model

Fig. 1 shows the involved parties and their interactions. SA²FE involves five parties: financial authority (FA), service provider (SP), base station (BS), edge server (ES) and user:

- 1) **FA:** The FA receives payment from users, and distributes tokens for service access. It also handles reward claims from BS and ES with valid tokens as proof of service.
- 2) **SP:** An SP owns a service and delegates it to ESs from various registered edge server infrastructure providers, delivering edge-based services to authorized users.
- 3) **BS:** The base station is the broker between users and ESs. It assists users within its range by discovering available

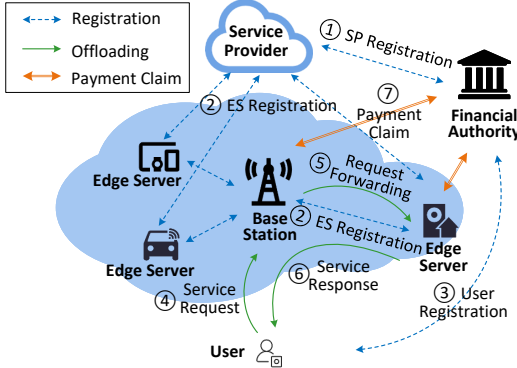


Fig. 1. SA²FE workflow. (1) SP registers to FA; (2) ES registers to SP and BS; (3) User gets tokens from FA; (4) User starts service request; (5) Request is forwarded to an ES; (6) User gets response; (7) BS and ES claim tokens. The workflow consists of three main phases: registration (steps (1)–(3)), offloading (steps (4)–(6)), and payment claim (step (7)).

ESs and offloading tasks of supported services. All communication between users and ESs will go through the BS.

- 4) **ES:** An ES provides services to users on behalf of SPs and may offer multiple services owned by different SPs.
- 5) **User:** A user requests task offloading for a service that she is subscribed to, and needs to provide payment (or proof of it) to utilize an edge-offloaded service.

We focus on a single BS but can be extended to multiple BSs managing different regions with shared information.

Interactions. Fig. 1 shows the interactions among these parties during an offloading process, with numbered steps as follows:

- 1) An SP registers service-related information with the FA.
- 2) An ES registers with an SP to obtain the service program (e.g. a virtual machine, container image or microservice), and service credentials to authenticate itself to users.
- 3) A user registers with the FA to deposit service pre-payments and obtain service access credentials (tokens).
- 4) When the user is within a BS which has connected ESs, the user requests offloading of her tasks from the BS.
- 5) A connected ES which is eligible to serve the request will be selected, and the BS forwards the service request to it.
- 6) The ES responds to the request, and starts the actual offloading process with the user.
- 7) The BS and ES claim pre-negotiated rewards from the FA with valid proof of service.

Existing work typically assumes full trust among the user, BS, and all ESs. For instance, the BS would neither intercept nor infer the user's service data or type and would ensure fair offloading to all ESs. ESs would provide services without intercepting user identity or unregistered service data. Users would not interfere the offloading process or engage in double spending for services. In practice, these assumptions would not always hold, especially in a democratized ecosystem where neither the user nor surrounding parties can be fully trusted.

B. Threat Model

We assume global parties (FA and SPs) will diligently adhere to the offloading protocol. This is because each global party may serve many users and stakeholders, and is commonly bound by reputation to perform honestly. In the mean time, local parties may deviate from the designated protocols to launch

active attacks, such as a BS of a small regional Internet Service Provider (ISP), or an ES from a local provider. Similarly, a user is not trusted to execute the protocol diligently.

A user may exhibit malicious behavior, such as expressing a preference for specific ESs, to disturb the fairness of service offloading. Furthermore, a user may target a specific ES to either perform reconnaissance attack in order to identify potential vulnerabilities of the ES and launch further attacks, or conduct targeted denial-of-service attacks to overwhelm target ES's resources. A user may also try to deceive both the BS and ES by engaging in double spending, utilizing the same payment to acquire multiple offloading services, possibly from different ESs. She may also use fraudulent authorization to acquire services without making valid payments.

Meanwhile, for a user, all other parties may possess a curiosity regarding the user's real identity and data for purposes such as data mining, targeted advertising, extracting personal information, and user tracking. Additionally, the user may want to hide her requested type of service from parties other than those required in purchasing and fulfilling the service, such as the BS and any ES that is not eligible to provide the service. Leaking the service type to untrusted parties compromises user privacy and poses security risks. For example, a medical offloading task exposes sensitive health information, while disclosure of the service type in financial transactions, location-based services, and personal preferences also poses privacy risks. The BS and ES may also exaggerate rewards, compromising FA integrity and user payments.

To summarize, We consider the following attack scenarios: (a) A malicious user may attempt to acquire services from ESs by double spending or forging payment proofs. (b) A malicious user may try to identify and request service from a specific ES, for instance, to disturb offloading fairness, perform reconnaissance of the ES's system, or launch denial-of-service attacks against the ES. (c) The BS and non-eligible ESs may be curious about users' real identity, data and the service type of the request. (d) The BS and ESs may be curious about a users' real identity, and the FA and SPs may want to link a user's identity with the time and location that she accesses a paid service. (e) The BS/ES may exaggerate the service it provided to get extra rewards from FA.

We assume a requested service is non-identifiable except by its service type, as many services, like video analytics, share similar traffic patterns despite differing tasks. There are also some studies on hiding traffic patterns from eavesdroppers, such as task partitioning or traffic padding [31].

C. Problem Statement

Let \mathcal{U} , \mathcal{S} , \mathcal{E} , \mathcal{B} be the set of users, set of services, set of ESs and the BS respectively. We consider an offloading scenario where a user $u \in \mathcal{U}$ offloads a task of service $s \in \mathcal{S}$ to an ES $e \in \mathcal{E}$ through the BS B . The user tries to conceal her identity from all other parties, and keep the service type hidden from the BS and non-eligible ESs. We require that neither the user nor the BS can "assign" an ES to serve a specific request; instead an eligible ES must be *randomly* selected for a specific request. This both deters user reconnaissance and other malicious behaviors against a

specific ES, and ensures fairness in offloading to promote community-wide sustainability and equal opportunities for all eligible ESs. The offloading service should only be provided when the user shows proof of payment, and the reward can only be claimed when the BS/ES shows proof of service, without double spending or exaggerated claiming.

Security goals. Our main security goals are as follows:

- 1) **Authenticated and authorized access:** Access to an ES-provided service is only limited to paid users of the service.
- 2) **Identity privacy:** A user's identity is kept confidential from other parties during and after the offloading process.
- 3) **Service data confidentiality:** The service data of a user is only accessible to an eligible ES providing the service.
- 4) **Service type confidentiality:** The BS and non-eligible ESs have no knowledge about the requested service type.
- 5) **Financial accountability:** A user cannot get more than the paid service using invalid or double-spent payment proof. BS or an ES cannot claim reward without fulfilling a request, or claim multiple rewards for one service fulfillment.
- 6) **Offloading fairness:** ESs eligible for selection by the user are assigned an equal probability of serving user requests, ensuring fairness in service allocation.

We focus on scenarios where the SP has performed profiling or testing of ESs to ensure that the capabilities and latency of ESs within the pool available to the user are similar. From the user's perspective, the quality of service or quality of experience provided by these ESs in the pool is equivalent or nearly identical. An out-of-band profiling phase enables the SP to assess the servicing capabilities of ESs [32]. This ensures fairness by simplifying the selection process while reducing user burden, avoiding biases caused by performance differences, and protecting ESs from being targeted by malicious users. Even in such scenarios, achieving fairness under our security goals is challenging, as it requires preserving service type confidentiality, ensuring efficient task allocation, and avoiding security risks or excessive overhead.

IV. SA²FE OVERVIEW

SA²FE operates in four phases: system initialization, registration, offloading, and payment claim. In **system initialization**, system parameters are independently initialized by all parties. In **registration**, an ES obtains permission to host a service from an SP, and then registers service information, encrypted as *cryptographic puzzles*, with a BS to serve offloading requests from local users. A user also registers with an SP and makes payment through an FA to obtain *blind tokens* for requesting the service. In **offloading**, a user requests an offloading service through the BS, solves the service-specific puzzles and randomly picks an ES capable of providing the service. The BS then forwards the (encrypted) request to the user-selected ES without knowing the service type. The request contains tokens specific to the requested service, such that the BS can check for any potential double spending (again without knowing the service type), and both BS and ES can later claim service payments from the FA with the tokens. In **payment claim**, the FA verifies the tokens submitted by the BS or ESs, and makes payment accordingly.

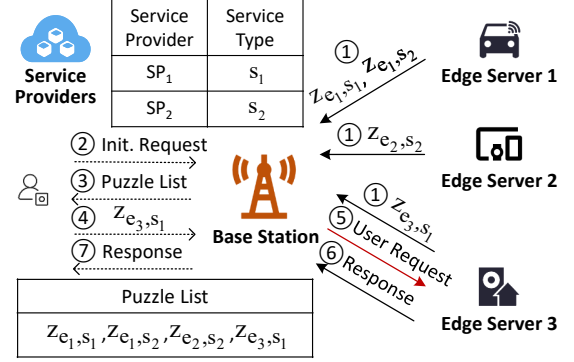


Fig. 2. Rerandomizable puzzle-based offloading example. Suppose there are two SPs offering two types of services, s_1 and s_2 , and three ESs e_1 , e_2 and e_3 attempting to assist the SPs in delivering services. e_1 provides both s_1 and s_2 , e_2 provides s_2 , and e_3 provides s_1 . Denote the puzzle of service s_j from edge server e_i as z_{e_i, s_j} . Suppose a user intends to use service s_1 . (1) ESs register service-related puzzles with the BS. (2) User initiates a request for an (unspecified) offloading service. (3) BS responds to user with a puzzle list. (4) User selects a puzzle z_{e_3, s_1} and returns it to the BS. (5) BS forwards the user's request to selected ES e_3 . (6) e_3 returns service response to the BS. (7) BS forwards the response to the user.

In SA²FE, the two key building blocks are: blind tokens for service access, and cryptographic puzzles for offloading.

Token-based service access. We develop a token-based service access scheme that ensures user anonymity. Our scheme maintains secrecy of the service type from the BS while allowing service type verification by the ESs. The token, specifically designed for authenticated and anonymous access to edge services, is blindly signed during user registration based on the blind signature scheme to preserve user privacy. A token contains a service-agnostic part for the BS, and a service-specific part for the ES signed by a service-related key. This allows the BS to check the token for potential double spending without knowing the service type, and the ES and FA to check the service type for potential token misuse during offloading and reward claiming respectively.

Puzzle-based offloading. To prevent the BS from learning a user's requested service type while still allowing forwarding the request to an eligible ES, we design a puzzle-based offloading process as shown in Fig. 2. The puzzle mechanism is essential for secure and efficient offloading in PEC environments. By preventing the BS from learning service type information and ensuring that users cannot target specific ESs, puzzles address threats like service type inference, malicious targeting, and fairness violations. Additionally, they enable a practical offloading process by eliminating the need for inefficient methods, such as random assignments or broadcasts by BS, which would otherwise increase complexity and resource consumption. Instead of registering plaintext service information at the BS, each ES will generate service-specific random puzzles that are indistinguishable from puzzles of other services. The puzzles can only be solved by users with each specific service's key obtained during user registration and payment. To ensure fair offloading, the BS sends all puzzles to the requesting user, who then solves the puzzles of its requested service, and then randomly picks one puzzle representing a random ES who can serve the service. The puzzle contains no identifiable information about the ES, ensuring that the user cannot identify or target a specific ES

during the selection (thus ensuring both fairness and protection of the ES). Further, after every service request, the BS randomizes all puzzles and permutes the puzzle list to ensure that puzzles from two requests are unlinkable.

A. Preliminary: Blind Signature

SA²FE makes use of a blind signature scheme as a building block, which we shall describe here for completeness.

Blind signature [33] is an unlinkable digital signature scheme that allows a signer to sign a message without knowing the message content. The algorithms are specified as follows:

- 1) BlindSetup(1^λ) \rightarrow (PK, SK): Given security parameter λ , it outputs the public key PK and secret key SK .
- 2) BlindMsg(PK, m, r) $\rightarrow m'$: Takes PK , message m , and random number r as input, outputs blinded message m' .
- 3) BlindSign(PK, SK, m') $\rightarrow s'$: Takes PK , SK , and blinded message m' as input, outputs signature s' .
- 4) UnblindSign(PK, s', r) $\rightarrow s$: Takes PK , s' , and random number r as input, outputs signature s for message m .
- 5) BlindVerify(PK, m, s) $\rightarrow \{0, 1\}$: Takes PK , m , and s as input, outputs 1 if s is valid for m , otherwise 0.

A secure blind signature scheme realizes two security properties: unforgeability and blindness [34]. Unforgeability ensures that only the signer can generate valid blind signatures. Blindness ensures that the signer cannot know the message content corresponding to the blind signature she has signed.

V. SA²FE DESIGN

In this section, we first design the puzzle primitive and present its two implementations. Table I lists SA²FE notations.

A. Puzzle Design

A **rerandomizable puzzle** is constructed for a specific solution. Anyone with the puzzle and the solution can verify that the solution is correct. It allows anyone with neither the solution nor any secret used when constructing the puzzle to rerandomize the puzzle without changing the solution.

Definition 1. A *rerandomizable puzzle* scheme consists of the following four algorithms:

- 1) PuzzleSetup(1^λ) \rightarrow *params*: Initialize puzzle parameters.
- 2) PuzzleGen(m) \rightarrow *puzzle*: Generate a *puzzle* given a solution message m .
- 3) PuzzleMatch($m, puzzle$) $\rightarrow \{0, 1\}$: Check if m is the solution to *puzzle*.
- 4) PuzzleRerandomize(*puzzle*) \rightarrow *new_puzzle*: Rerandomize *puzzle* without changing the solution m , such that *new_puzzle* is unlinkable to *puzzle*. \square

The following properties must be fulfilled: (a) Correctness: PuzzleMatch($m, \text{PuzzleGen}(m) = 1$ for any m ; (b) Soundness: $\Pr[\text{PuzzleMatch}(\hat{m}, \text{PuzzleGen}(m)) = 1] \approx 0$ for $\hat{m} \neq m$; (c) Indistinguishability: it is computationally hard to distinguish $\text{PuzzleGen}(m)$ from $\text{PuzzleGen}(\hat{m})$ for any $m \neq \hat{m}$; (d) Unlinkability: given a *puzzle*, a *new_puzzle* can be generated such that $\text{PuzzleMatch}(m, \text{new_puzzle}) = 1$ and *new_puzzle* is unlinkable to *puzzle* for any m .

In the following, we propose two puzzle implementations based on bilinear map and universal re-encryption respectively to realize the rerandomizable puzzle primitive.

TABLE I
NOTATION TABLE

Symbol	Definition	Symbol	Definition
λ	Security parameter	k_s	Service key
pk_p, sk_p	FA public & secret key	s_type	Service type
pk_s, sk_s	SP public & secret key	s_alg	Service program
m_1, m_2	Random messages	Z_{ID_u}	User puzzle list
p_map	Puzzle mapping table		

Puzzle based on bilinear map. Let \mathbb{G}_1 , \mathbb{G}_2 and \mathbb{G}_T denote three cyclic groups of a prime order p , and g_1 and g_2 be the generators of group \mathbb{G}_1 and \mathbb{G}_2 respectively. A bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ satisfying bilinearity, computability, and non-degeneracy can be used to construct the bilinear-based rerandomizable puzzle as follows.

- 1) PuzzleSetup(1^λ) \rightarrow *params*: Let *params* = (g_1, g_2).
- 2) PuzzleGen(m) \rightarrow *puzzle*: Generate random factor $r \in \mathbb{Z}_p^*$ where $r \bmod m = 0$. Then output *puzzle* = ($z_{[1]}, z_{[2]}$), where $z_{[1]} = g_1^{r/m}$ and $z_{[2]} = g_2^r$.
- 3) PuzzleMatch($m, puzzle$) $\rightarrow \{0, 1\}$: Check if $e(g_1^m, z_{[1]}) = e(g_1, z_{[2]})$.
- 4) PuzzleRerandomize(*puzzle*) \rightarrow *new_puzzle*: Output *new_puzzle* = ($(z_{[1]})^{r'}, (z_{[2]})^{r'}$) with random $r' \in \mathbb{Z}_p^*$.

Puzzle based on universal re-encryption. ElGamal encryption [35] is an asymmetric key encryption scheme for public-key cryptography. With the public key, any ciphertext can be re-encrypted into an unrelated ciphertext. The universal re-encryption scheme [36] hides the public key by appending a second ElGamal ciphertext encrypting the integer 1. Leveraging ElGamal's algebraic homomorphism, the second ciphertext can re-encrypt the first without exposing the public key. The universal re-encryption-based puzzle is constructed as follows.

- 1) PuzzleSetup(1^λ) \rightarrow (x, y): Output ($x, y = g^x$), where $x \in \mathbb{Z}_q$, g is a generator for a group \mathbb{G} with order q .
- 2) PuzzleGen(m) \rightarrow *puzzle*: Generate random factor $r = (r_0, r_1) \in \mathbb{Z}_q^2$. Then output *puzzle* = $[(\alpha_0, \beta_0); (\alpha_1, \beta_1)] = [(my^{r_0}, g^{r_0}); (y^{r_1}, g^{r_1})]$.
- 3) PuzzleMatch($m, puzzle$) $\rightarrow \{0, 1\}$: Verify $\alpha_0, \beta_0, \alpha_1, \beta_1 \in \mathbb{G}$, return 0 if invalid. Compute $m_0 = \alpha_0/\beta_0^x$ and $m_1 = \alpha_1/\beta_1^x$. If $m_1 = 1$, return 1 if $m_0 = m$.
- 4) PuzzleRerandomize(*puzzle*) \rightarrow *new_puzzle*: Output $[(\alpha'_0, \beta'_0); (\alpha'_1, \beta'_1)] = [(\alpha_0\alpha_1^{r'_0}, \beta_0\beta_1^{r'_0}); (\alpha_1^{r'_1}, \beta_1^{r'_1})]$ with random factor $r' = (r'_0, r'_1) \in \mathbb{Z}_q^2$.

These two designs are based on different assumptions and have different overheads. The bilinear map puzzle and the universal re-encryption puzzle are based on Decisional Bilinear Diffie-Hellman assumption (DBDH) [37] and Decisional Diffie-Hellman assumption (DDH) [38], respectively. We evaluate the overhead of these two designs in the Section VII.

Theorem 1. The bilinear map-based puzzle and the universal re-encryption-based puzzle satisfy the properties of correctness, soundness, indistinguishability, and unlinkability as defined for a randomizable puzzle scheme.

Proof. We provide proofs for each property for both the bilinear map-based and universal re-encryption-based puzzles.

Correctness: The correctness of the bilinear map-based puzzle follows from $e(g_1^m, g_2^{r/m}) = e(g_1, g_2^r)$. For the universal re-encryption-based puzzle, correctness follows from $\alpha_1/\beta_1^x = y^{r_1}/g^{r_1x} = 1$ and $\alpha_0/\beta_0^x = my^{r_0}/g^{r_0x} = m$.

Algorithm 1: System Initialization

At Service Provider out-of-band from the SP;
1 BlindSetup(1^λ) \rightarrow (pk_s, sk_s); **At Financial Authority**
2 SymKeySetup(1^λ) $\rightarrow k_s$; 4 BlindSetup(1^λ) \rightarrow (pk_p, sk_p);
At User 5 PuzzleSetup(1^λ) $\rightarrow params$.
3 Get blind signature public key pk_s

Soundness: The soundness of the bilinear map-based puzzle holds because, for any $\hat{m} \neq m$, $e(g_1^{\hat{m}}, g_2^{r/m}) \neq e(g_1, g_2^r)$. For the universal re-encryption-based puzzle, soundness is ensured since $\hat{m}y^{r_0}/g^{r_0x} \neq m$ for any $\hat{m} \neq m$.

Indistinguishability: The bilinear map-based puzzle's indistinguishability relies on the DBDH assumption, ensuring $(g_2^{r/m}, g_2^r)$ and $(g_2^{\hat{r}/\hat{m}}, g_2^{\hat{r}})$ with $e(g_1^m, g_2^{r/m}) = e(g_1, g_2^r)$ and $e(g_1^{\hat{m}}, g_2^{\hat{r}/\hat{m}}) = e(g_1, g_2^{\hat{r}})$ are indistinguishable. For the universal re-encryption-based puzzle, it derives from the DDH assumption, ensuring $[(m(g^x)^{r_0}, g^{r_0}); ((g^x)^{r_1}, g^{r_1})]$ and $[(\hat{m}(g^x)^{r_0}, g^{r_0}); ((g^x)^{r_1}, g^{r_1})]$ are indistinguishable.

Unlinkability: The bilinear map-based puzzle's unlinkability also relies on the DBDH assumption, ensuring that $(g_2^{r/m}, g_2^r)$ and $(g_2^{rr'/m}, g_2^{rr'})$ with $e(g_1^m, g_2^{r/m}) = e(g_1, g_2^r)$ and $e(g_1^m, g_2^{rr'/m}) = e(g_1, g_2^{rr'})$ remain indistinguishable, ensuring unlinkability. For the universal re-encryption-based puzzle, unlinkability similarly derives from the DDH assumption, ensuring that $[(m(g^x)^{r_0}, g^{r_0}); ((g^x)^{r_1}, g^{r_1})]$ and $[(m(g^x)^{r_0}, g^{r_0}); ((g^x)^{r_1}, g^{r_1})]$ are indistinguishable, ensuring unlinkability between the original puzzle and the rerandomized puzzle. \square

We will integrate this puzzle primitive into the offloading scheme, enabling the ES to register at the BS and allowing users to randomly select an indistinguishable puzzle from the BS's puzzle list to ensure fairness in offloading.

B. System Initialization

The system initialization phase initializes all parameters and components required for the system, as shown in Algorithm 1. **Service setup (line 2).** The SP sets up the service key k_s that will be used for symmetric encryption of service data. A symmetric encryption scheme has three algorithms, SymKeySetup, SymEnc and SymDec. We employ symmetric encryption for the serviced data to reduce the overhead of encrypting and decrypting. Alternative encryption schemes can be used if they are compatible with the service data. Due to the lightweight nature of our framework, the service key can be efficiently updated periodically based on security requirements or manually rotated upon detecting anomalies.

Blind signature setup (lines 1, 3–4). Blind signature [33] is employed to preserve user anonymity during token usage and maintain confidentiality of the service type from the BS, while still enabling service type verification by the ES and FA.

Each token includes a service-agnostic part for the BS and a service-specific part for the ES and FA, supported by blind signature setups from the SP (line 1) and FA (line 4). Regarding the service-specific part, the service blind signature secret key sk_s is securely kept confidential by the FA after the SP registers with it, as will be shown in Algorithm 2. And the service blind signature public key pk_s is only accessible to authorized users and ESs who can access the service s (line 3). Regarding the service-agnostic part, the FA invokes

Algorithm 2: Registration

/ SP registration */* 11 UnblindSign(pk_s, sig'_2, r) $\rightarrow sig_2$;
At Service Provider 12 $token = (m_1, sig_1; m_2, sig_2)$;
1 Register with FA by sending k_s and */* ES reg. to SP */*
(pk_s, sk_s) to the FA. **At Edge Server**
/ Token registration */* 13 Send (reg_info, s_type) to SP;
At User **At Service Provider**
2 Select two random messages m_1 14 **if ES eligibility is verified then**
and m_2 , and a random number r ; 15 | Send (k_s, pk_s, s_alg) to ES;
3 BlindMsg(pk_p, m_1, r) $\rightarrow m'_1$; */* ES reg. to BS */*
4 BlindMsg(pk_s, m_2, r) $\rightarrow m'_2$; **At Edge Server**
5 Send request (s_type, m'_1, m'_2 , 16 **for registered service s do**
payment) to the FA; 17 | PuzzleGen($h(k_s)$) $\rightarrow puzzle$;
At Financial Authority 18 | Send ($puzzle, ID_{BS}$) to BS;
6 **if registration request is valid then** **At Base Station**
7 | BlindSign(sk_p, m'_1) $\rightarrow sig'_1$; 19 **while puzzle from ID_{ES} do**
8 | BlindSign(sk_s, m'_2) $\rightarrow sig'_2$; 20 | Store $puzzle$ in puzzle list Z ;
9 | Send (sig'_1, sig'_2, k_s) to user; 21 | Insert ($puzzle, ID_{ES}$) into a
At User mapping table p_map .
10 UnblindSign(pk_p, sig'_1, r) $\rightarrow sig_1$;

BlindSetup and the public key pk_p is made publicly available, allowing the BS to verify the service-agnostic part of tokens. **Rerandomizable puzzle setup (line 5).** The FA sets up the parameters of rerandomizable puzzle. For puzzle based on the bilinear map, both g_1 and g_2 can be published. For puzzle based on universal re-encryption, only y can be made public, while the corresponding x is obtained by authorized users.

C. Registration

Registration phase (steps (1)–(3) in Fig. 1) is in Algorithm 2. **SP registration (line 1).** An SP registers the service symmetric key k_s , the service blind signature public key pk_s and secret key sk_s with the FA. Then the FA can issue and verify the validity of service tokens for service s by using these keys. **Token registration (lines 2–12).** To access a service s , a user first needs to acquire tokens for s from the FA, which needs to include both a service-agnostic part for the BS, and a service-specific part for the ES and FA. To request a token for service s , the user selects random messages m_1, m_2 , and a random factor r , then blinds m_1 and m_2 using BlindMsg to generate blind messages (lines 2–4). Then the user sends blinded messages m'_1 and m'_2 along with payment information to the FA (line 5). After verifying the payment, the FA signs m'_1 and m'_2 by invoking BlindSign. Then the FA sends the blinded signatures and the service key k_s back to the user (lines 6–9). After invoking UnblindSign, the user gets a valid $token$ in the format of $(m_1, sig_1; m_2, sig_2)$ (lines 10–12).

ES registration to SP (lines 13–15). The ES first generates a registration request and sends it to the SP. The SP checks service type and verifies if the ES with reg_info can provide the service s . If the ES is eligible, the SP sends back service key k_s , blind signature key pk_s , and service program s_alg .

ES registration to BS (lines 16–21). After receiving the service information, each ES registers with the BS as a candidate to provide service. This process enables ES discovery and equips users with information for fair ES selection. For registered service s , the ES generates a puzzle by invoking PuzzleGen($h(k_s)$), where $h(\cdot)$ is a one-way hash function to protect the service key. Upon receiving the puzzles from ESs, the BS stores puzzles in list Z and creates a mapping table p_map associating each puzzle with ES identity ID_{ES} .

To adapt to varying ES capabilities, fairness can be extended by allowing higher-capability ESs to register multiple puzzles,

Algorithm 3: Offloading

At User
1 Generate offloading request ($token$, ID_{BS}) and send it to BS;
At Base Station
2 Form puzzle list Z_{ID_u} with the latest versions for user ID_u ;
3 **for** $z \in Z_{ID_u}$ **do**
4 PuzzleRerandomize(z) $\rightarrow z'$;
5 Replace z with z' ;
6 Record (z' , ID_{ES}) in p_map ;
7 Permute Z_{ID_u} to a new puzzle list Z'_{ID_u} and send it to user;
At User
8 Candidate puzzle list $Z_c = \emptyset$;
9 **for** $z \in Z'_{ID_u}$ **and**
10 PuzzleMatch($h(k_s), z$) = 1 **do**
11 $Z_c = Z_c \cup \{z\}$;
12 Randomly pick a puzzle $z_u \in Z_c$;
13 Send (z_u , $ct = \text{SymEnc}(k_s, (s_type, data))$) to BS;

At Base Station
14 **if** BlindVerify(pk_p, m_1, sig_1) = 1 **and** token is unseen **then**
15 **if** $z_u \in Z'_{ID_u}$ and none of puzzles in Z'_{ID_u} have been used **then**
16 Send ($token$, ct) to ES according to p_map ;
At Edge Server
17 **for** $\forall pk_s$ held by the ES **do**
18 **if** BlindVerify(pk_s, m_2, sig_2) = 1 **then**
19 SymDec(k_s, ct) $\rightarrow data$;
20 Send $resp = \text{SymEnc}(k_s, s_alg(data))$ to BS;
break;
At Base Station
21 Forward the $resp$ to user;
At User
22 SymDec($k_s, resp$) $\rightarrow resp_data$.

increasing their selection likelihood proportionally to their capacity while maintaining efficiency and security.

D. Offloading

Algorithm 3 shows the detailed offloading phase, which corresponds to steps (4)–(6) in Fig. 1.

The user initiates offloading by sending a request to the BS (line 1). Upon receiving the *init_request* from the user with ID_U , the BS performs the following actions (lines 2–7): the BS first constructs a puzzle list Z_{ID_u} that contains all the latest version puzzles. Then the BS re-randomizes the puzzles $z \in Z_{ID_u}$ by invoking $\text{PuzzleRerandomize}(z) \rightarrow z'$ and replaces z with z' . Also, the BS records (z' , ID_{ES}) in p_map . The BS sends the permuted puzzle list Z'_{ID_u} to the user. The BS re-randomizes and permutes puzzles to ensure fairness, preventing users from targeting specific ESs. While acting as a man-in-the-middle, it cannot infer service types, with protocol adherence incentivized by its reliance on reputation.

The user, upon receiving Z'_{ID_u} , proceeds with the following steps (lines 8–12): for each puzzle received from the BS, the user matches it with the service key for the desired service, constructing a sub-list Z_c of matching puzzles. The user then randomly selects one puzzle z_u from Z_c . The user encrypts request *data* with service key k_s to get the ciphertext *ct*, and constructs a message (z_u , *ct*) which is then sent to the BS.

Upon receiving the user's offloading request, the BS performs the following steps (lines 13–15): the BS checks the validity of the service-agnostic part of the token (m_1, sig_1) by invoking *BlindVerify* and ensuring full token has not been used before. The BS validates z_u by confirming its presence in the unique puzzle list Z'_{ID_u} provided to user ID_U and ensuring that none of the puzzles in Z'_{ID_u} have been used before. The BS then finds the ES corresponding to z_u in its mapping table p_map , and forwards ($token, ct$) to the ES.

Then the ES verifies the service-specific part of the token by invoking *BlindVerify*(pk_s, m_2, sig_2). For ESs offering multiple services, they need to check the service blind signature public key, pk_s , associated with each service type to find the corresponding service key, k_s (lines 16–17). If the token check passes, the ES decrypts *ct* (line 18) and continues generating response data for the user using service algorithm s_alg on

Algorithm 4: Payment Claim

1 **while** ($s_type, token$) from ES **do**
2 **if** BlindVerify(pk_s, m_2, sig_2) = 1 **and** token valid for s_type **then** FA pays to ES and SP;
3 **while** ($token$) from BS **do**
4 **if** BlindVerify(pk_p, m_1, sig_1) = 1 **and** token not double spent **then** FA pays to BS.

data. The ES encrypts the response data with k_s and sends it to the BS (line 19). The BS forwards the encrypted *resp* to the user (line 21), allowing the user to decrypt it with k_s (line 22). The user and the ES then engage in actual service offloading through the BS until the offloading request is fulfilled.

E. Payment Claim

The payment claim process in Fig. 1 step (7) is shown in Algorithm 4. The process is the same for a BS or an ES, except that the public key used to verify the blind signature is different, and for ES the checking needs to additionally verify s_type . Upon receiving a token claim request, the FA first checks whether the token has been double spent. It then proceeds to verify the token's validity by invoking the function *BlindVerify*. Upon successful token verification, the FA pays the corresponding tokens to the SP, BS, and ES as per the established contract, ensuring accountability for token claims.

In practice, beyond presenting a valid *token* as payment proof, the BS and ES can incorporate other types of proof of service to claim rewards. For instance, they can utilize existing edge service verification schemes that leverage cryptography to generate tamper-proof service proofs [39].

VI. SECURITY ANALYSIS**A. Informal Security Analysis**

Malicious user. Upon receiving the user's offloading request, the BS validates the token's service-agnostic part for FA signature and checks both token parts for prior use. Only when the check on (m_1, sig_1) returns valid and the full token has not been seen before, will the BS proceed to construct a puzzle list and share it with the user. Full token checking ensures that the user cannot reuse a forged token, for instance, by combining a low-priced BS service-agnostic part with a high-priced service-specific part to double spend the token. The BS's re-randomization and permutation of the puzzle list ensure the unlinkability within one round and different rounds of the offloading, so that the user cannot identify a specific ES to disturb the fairness of the offloading process. Stale puzzle submissions are discarded, thwarting puzzle replay attacks.

Curious BS on user service request. SA²FE prevents a curious BS from inferring a user's service type by limiting access to sensitive information. The BS can only verify the service-agnostic part of the token, which reveals no service details. User data is encrypted with the service symmetric key, preventing access to the service type or request data. Additionally, puzzles from ESs are indistinguishable and disclose no service-related information. Since the user performs puzzle matching and selection, the BS cannot infer the service type from the list of eligible ESs or from specific ES involvement. **Curious FA, SP, BS, and ES on user identity.** Due to the blind signature's blindness properties, no one can link the token in a service request to the user's real identity.

Malicious BS and ES on payment claim. If the BS or an ES exaggerates the provided service for extra rewards, the FA will detect invalid or double-spent tokens and reject the claim.

For a puzzle list of n puzzles, the probability of a malicious user identifying a specific ES is $\frac{1}{n}$. The attack success probability of other attacks within our threat model are negligible unless they violate fundamental cryptographic assumptions.

B. Formal Security Analysis

We next formally analyze SA²FE's security based on the UC framework. The UC is a widely used simulation-based cryptographic framework for modular security analysis in diverse scenarios, including blockchain [40], federated learning [41], and quantum key distribution [42]. It guarantees security even when a secure protocol is composed with an arbitrary set of protocols [43]. The definition of UC-security is as follows:

Definition 2. UC-security [43]. Given a security parameter λ , an ideal functionality \mathcal{F} and a real world protocol π , we say that π securely realizes \mathcal{F} if for any probabilistic polynomial time (PPT) adversary \mathcal{A} , there exists a PPT simulator \mathcal{S} such that for any PPT environment \mathcal{Z} , we have

$$\text{IDEAL}_{\mathcal{F}, \mathcal{S}, \mathcal{Z}} \stackrel{c}{=} \text{REAL}_{\pi, \mathcal{A}, \mathcal{Z}}.$$

The $\stackrel{c}{=}$ denotes computational indistinguishable. \square

We denote the ideal functionality of SA²FE as $\mathcal{F}_{\text{SA}^2\text{FE}} = \langle \mathcal{F}_{\text{register}}, \mathcal{F}_{\text{offload}}, \mathcal{F}_{\text{claim}}, \mathcal{F}_{\text{sig}}, \mathcal{F}_{\text{smt}} \rangle$. $\mathcal{F}_{\text{register}}$ is the ideal functionality of registration phase. $\mathcal{F}_{\text{offload}}$ models the offloading phase. $\mathcal{F}_{\text{claim}}$ models the token claim process. Two helper ideal functionalities \mathcal{F}_{sig} and \mathcal{F}_{smt} [43] are used to model the digital signature and the secure message transmission channel. Following UC framework, we assume that each party interacting with the ideal functionalities has a unique identifier, and consider a static corruption model where the adversary can corrupt parties at the beginning of the protocol.

The ideal functionality $\mathcal{F}_{\text{SA}^2\text{FE}}$ maintains the internal states in three tables, T_s , T_p and T_t , to ensure consistency of the real world and the ideal world. T_s consists of entries in the format of $(\text{spid}, \text{sname}, \text{sdata}, \text{esid}, \text{bsid})$ about the service. The spid denotes the SP identity, sname represents the service name, sdata contains the content or data associated with the service, and esid and bsid uniquely identify an ES and a BS respectively. T_p contains puzzle information in the format of $(\text{puzzle}_{\text{ideal}}, \text{spid}, \text{sname}, \text{ver}, f_p, \text{esid}, \text{bsid})$. The $\text{puzzle}_{\text{ideal}}$ is a puzzle in the ideal world. ver is a number that indicates the puzzle's version that identifies the set of puzzles corresponding to a user's request. The puzzles generated in the same batch have the same version number. $f_p \in \{\text{unused}, \text{used}\}$ is a flag indicating whether the puzzle has been received by the BS from a user, esid is the ES identity that the puzzle corresponds to, and bsid is the BS identity that the esid is registered with. T_t contains the token information in the format of $(\text{token}_{\text{ideal}}, \text{spid}, \text{sname}, (\text{esid}, f_{es}), (\text{bsid}, f_{bs}))$. The $\text{token}_{\text{ideal}}$ is a string that indicates the token in the ideal world. esid is the ES identity that received the token and f_{es} is the flag that indicates the status of the token. f_{es} has three options: fresh for a newly initialized, unused token, unclaimed for token received but not yet claimed, and claimed for token already claimed. (bsid, f_{bs}) is similarly defined for a BS.

Functionality $\mathcal{F}_{\text{register}}$

Service provider registration

- 1) Upon receiving $(\text{register}, \text{spid}, \text{sname}, \text{sdata})$ from the SP, $\mathcal{F}_{\text{register}}$ adds $t_s = (\text{spid}, \text{sname}, \text{sdata}, \perp, \perp)$ to T_s . If the t_s is already in T_s , then $\mathcal{F}_{\text{register}}$ returns t_s to the SP.

Edge server registration

- 1) Upon receiving $(\text{register}, \text{spid}, \text{sname}, \text{esid})$ from ES, $\mathcal{F}_{\text{register}}$ checks if T_s has an entry $t_s = (\text{spid}, \text{sname}, \cdot, \text{esid}, \cdot)$. If yes, $\mathcal{F}_{\text{register}}$ returns t_s to esid , and forwards (exist, t_s) to S . Otherwise, $\mathcal{F}_{\text{register}}$ sends a message $(\text{register}, \text{spid}, \text{esid})$ to the SP with spid . If the SP responds with "allow", then $\mathcal{F}_{\text{register}}$ creates an entry $(\text{spid}, \text{sname}, \cdot, \text{esid}, \cdot)$ in T_s and forwards $(\text{successReg}, (\text{spid}, \text{sname}, \text{esid}))$ to esid and S . Otherwise, $\mathcal{F}_{\text{register}}$ returns "fail" to esid and forwards $(\text{failReg}, (\text{spid}, \text{sname}, \text{esid}))$ to S .
- 2) Upon receiving $(\text{register}, t_p = (\cdot, \text{spid}, \text{sname}, 0, \text{unused}, \text{esid}, \text{bsid}), \text{bsid})$ from ES, $\mathcal{F}_{\text{register}}$ forwards $(\text{register}, \text{esid}, \text{bsid})$ to BS with bsid . If BS responds with "allow", then $\mathcal{F}_{\text{register}}$ updates the entry $(\text{spid}, \text{sname}, \cdot, \text{esid}, \text{bsid})$ in T_s and forwards $(\text{successReg}, (\text{spid}, \text{sname}, \text{bsid}, \text{esid}))$ to esid and S . Otherwise, $\mathcal{F}_{\text{register}}$ returns "fail" to esid and forwards $(\text{failReg}, (\text{spid}, \text{sname}, \text{bsid}, \text{esid}))$ to S .
- 3) $\mathcal{F}_{\text{register}}$ adds $t_p = (\text{puzzle}_{\text{ideal}}, \text{spid}, \text{sname}, 0, \text{unused}, \text{esid}, \text{bsid})$ to T_p , and forwards (newReg, t_p) to S .

User registration

- 1) Upon receiving $(\text{register}, \text{spid}, \text{sname}, \text{payment})$ from user, $\mathcal{F}_{\text{register}}$ sends a message $(\text{register}, \text{spid}, \text{sname}, \text{payment})$ to FA.
- 2) If FA returns "allow", $\mathcal{F}_{\text{register}}$ adds $t_t = (\text{token}_{\text{ideal}}, \text{spid}, \text{sname}, (\perp, \text{fresh}), (\perp, \text{fresh}))$ in T_t . Then $\mathcal{F}_{\text{register}}$ sends the t_t to the user and forwards (newReg, t_t) to S .
- 3) Otherwise, $\mathcal{F}_{\text{register}}$ sends "fail" to user and sends $(\text{failReg}, t_t)$ to S .

Fig. 3. Ideal functionality for registration.

Registration. The ideal functionality $\mathcal{F}_{\text{register}}$, shown in Fig. 3, handles registration for the SP, BS, and ES by creating entries for services, puzzles, and tokens, ensuring validity and freshness while coordinating with other parties.

Offloading. The ideal functionality shown in Fig. 4 models the offloading process between the user, BS and ES. The functionality validates tokens and checks if selected puzzles are valid entries in T_p . Invalid tokens or puzzles result in a failure message to the user and a notification to S . For valid requests, it updates the status of tokens and puzzles, marks puzzles as used, changes the status of tokens, and manages mappings in T_t and T_p . Finally, $\mathcal{F}_{\text{offload}}$ forwards the service response M_{resp} to the user, BS, and S .

Payment Claim. Fig. 5 shows the payment claim ideal functionality, where $\mathcal{F}_{\text{claim}}$ verifies token validity, prevents double spending by checking f_{bs} or f_{es} status, rewards bsid or esid , and updates f_{bs} or f_{es} to mark the token as claimed.

Theorem 2. Let \mathcal{A} and \mathcal{S} be a PPT adversary and a simulator in the real world and the ideal world, respectively. SA²FE securely realizes $\mathcal{F}_{\text{SA}^2\text{FE}}$ for any PPT environment \mathcal{Z} .

Proof. We design a series of games, where each game differs slightly from the previous one but remains indistinguishable from the view of the PPT environment \mathcal{Z} .

Game 0: This is the real world protocol SA²FE that interacts directly with the environment \mathcal{Z} and adversary \mathcal{A} .

Game 1: This game is identical to Game 0 except that the real-world communication channel is replaced by \mathcal{F}_{smt} .

Lemma 1. For any \mathcal{A} and \mathcal{Z} , there exists an \mathcal{S} such that the view of \mathcal{Z} in Game 1 is indistinguishable from its view in Game 0, i.e., $\text{Exec}_{\text{Game0}, \mathcal{Z}} \approx \text{Exec}_{\text{Game1}, \mathcal{Z}}$.

Proof. \mathcal{S} can run \mathcal{S}_{smt} for \mathcal{F}_{smt} to achieve the indistinguishability between the real world and ideal world. \square

Functionality $\mathcal{F}_{\text{offload}}$

- 1) Upon receiving a request $(\text{offRequest}, \text{token}_{\text{ideal}}, \text{bsid})$ from the user, $\mathcal{F}_{\text{offload}}$ constructs a p_list which includes all the puzzle filed of the puzzles with $(\cdot, \cdot, \cdot, \text{ver}_{\text{newest}}, \cdot, \cdot, \text{bsid})$ in the T_p .
- 2) $\mathcal{F}_{\text{offload}}$ re-randomizes each puzzle in p_list by generating a new puzzle string $\text{puzzle}_{\text{new}}$, and creates a new entry $t_p = (\text{puzzle}_{\text{new}}, \text{spid}, \text{sname}, \text{ver}_{\text{newest}} + 1, \text{unused}, \text{esid}, \text{bsid})$ in T_p . $\mathcal{F}_{\text{offload}}$ sends $(\text{randomizedPuzzle}, \text{puzzle}_{\text{ideal}}, \text{puzzle}_{\text{new}})$ to \mathcal{S} .
- 3) $\mathcal{F}_{\text{offload}}$ constructs a p_list' for user by collecting all puzzles with $\text{ver}_{\text{newest}} + 1$ in a random order. If the user is malicious, $\mathcal{F}_{\text{offload}}$ sends p_list' to bsid and forwards $(\text{offStart}, \text{token}, p_list')$ to \mathcal{S} . If the user is honest, $\mathcal{F}_{\text{offload}}$ associates each puzzle in the p_list' with its corresponding $(\text{spid}, \text{sname})$ and sends the list to the user. $\mathcal{F}_{\text{offload}}$ stores the mapping between uid and the corresponding ver^* .
- 4) Upon receiving response from the user with $(\text{puzzle}'_{\text{ideal}}, M_{\text{data}})$, $\mathcal{F}_{\text{offload}}$ sends $(\text{newRequest}, \text{puzzle}'_{\text{ideal}}, M_{\text{data}})$ to \mathcal{S} . $\mathcal{F}_{\text{offload}}$ sends $(\text{userAbort}, \text{uid})$ to \mathcal{S} when there is no response.
- 5) $\mathcal{F}_{\text{offload}}$ checks if there is an entry $t_t = (\text{token}_{\text{ideal}}, \text{spid}, \text{sname}, (\perp, \text{fresh}), (\perp, \text{fresh}))$ in T_t . If no such t_t exist, $\mathcal{F}_{\text{offload}}$ returns "fail" to user and forwards $(\text{invalidToken}, \text{token}_{\text{ideal}}, \text{bsid})$ to \mathcal{S} .
- 6) If there is such an entry t_t , then $\mathcal{F}_{\text{offload}}$ sends $(\text{newRequest}, \text{puzzle}'_{\text{ideal}}, M_{\text{data}})$ to bsid and \mathcal{S} . $\mathcal{F}_{\text{offload}}$ verifies the validity of $\text{puzzle}'_{\text{ideal}}$ within p_list' . If not, $\mathcal{F}_{\text{offload}}$ returns "fail" to user and forwards $(\text{invalidPuzzle}, \text{puzzle}'_{\text{ideal}}, M_{\text{data}})$ to \mathcal{S} .
- 7) If $\text{puzzle}'_{\text{ideal}}$ is valid and f_p is unused, $\mathcal{F}_{\text{offload}}$ retrieves the esid corresponding to $\text{puzzle}'_{\text{ideal}}$ and updates the entries with ver^* to $(\cdot, \cdot, \cdot, \text{ver}^*, \text{used}, \text{esid}, \text{bsid})$. Then $\mathcal{F}_{\text{offload}}$ updates the T_t with $(\text{token}_{\text{ideal}}, \text{spid}, \text{sname}, (\text{esid}, \text{fresh}), (\text{bsid}, \text{unclaimed}))$. Otherwise, $\mathcal{F}_{\text{offload}}$ returns "fail" to user. And $\mathcal{F}_{\text{offload}}$ forwards $(\text{invalidPuzzle}, \text{puzzle}'_{\text{ideal}}, M_{\text{data}})$ to \mathcal{S} .
- 8) $\mathcal{F}_{\text{offload}}$ sends $(\text{token}_{\text{ideal}}, M_{\text{data}})$ to esid and sends $(\text{offToES}, \text{token}_{\text{ideal}}, \text{esid}, M_{\text{data}})$ to \mathcal{S} . $\mathcal{F}_{\text{offload}}$ updates the T_t with $(\text{token}_{\text{ideal}}, \text{spid}, \text{sname}, (\text{esid}, \text{unclaimed}), (\text{bsid}, \text{unclaimed}))$.
- 9) On receiving $(\text{token}_{\text{ideal}}, M_{\text{data}})$ from $\mathcal{F}_{\text{offload}}$, esid retrieves s_{data} from $\mathcal{F}_{\text{offload}}$ and sends $M_{\text{resp}} \leftarrow s_{\text{data}}(M_{\text{data}})$ to $\mathcal{F}_{\text{offload}}$.
- 10) $\mathcal{F}_{\text{offload}}$ forwards the response M_{resp} to the user, bsid and \mathcal{S} .

Fig. 4. Ideal functionality for offloading.

Functionality $\mathcal{F}_{\text{claim}}$

- 1) Upon receiving $(\text{claimRequest}, \text{bsid}, \text{token}_{\text{ideal}})$ from bsid , $\mathcal{F}_{\text{claim}}$ checks if there is an entry $t_t = (\text{token}_{\text{ideal}}, \cdot, \cdot, (\cdot, \cdot), (\text{bsid}, \text{unclaimed}))$ in T_t . If no such t_t exist, then $\mathcal{F}_{\text{claim}}$ returns "fail" to bsid and forwards $(\text{invalidToken}, \text{token}_{\text{ideal}}, \text{bsid})$ to \mathcal{S} . Otherwise, $\mathcal{F}_{\text{claim}}$ sets the f_{bs} of t_t to claimed and returns "success" to bsid . Also, $\mathcal{F}_{\text{claim}}$ forwards $(\text{successClaimed}, \text{token}_{\text{ideal}}, \text{bsid})$ to \mathcal{S} .
- 2) Upon receiving $(\text{claimRequest}, \text{esid}, \text{spid}, \text{sname}, \text{token}_{\text{ideal}})$ from esid , $\mathcal{F}_{\text{claim}}$ checks if there is an entry $t_t = (\text{token}_{\text{ideal}}, \text{spid}, \text{sname}, (\text{esid}, \text{unclaimed}), (\cdot, \cdot))$ in T_t . If not, $\mathcal{F}_{\text{claim}}$ returns "fail" to esid and forwards $(\text{invalidToken}, \text{token}_{\text{ideal}}, \text{esid})$ to \mathcal{S} . Otherwise, $\mathcal{F}_{\text{claim}}$ sets the f_{es} of t_t to claimed and returns "success" to esid . Also, $\mathcal{F}_{\text{claim}}$ forwards $(\text{successClaimed}, \text{token}_{\text{ideal}}, \text{esid})$ to \mathcal{S} .

Fig. 5. Ideal functionality for payment claim.

Game 2: Let \mathcal{S} have access to the output of both honest parties and the adversary \mathcal{A} . Then \mathcal{S} tries to simulate the protocol with the help of $\mathcal{F}_{\text{SA}^2\text{FE}}$. In the real world, \mathcal{A} can corrupt the entities. Subsequently, all incoming and outgoing messages of the corrupted party go through \mathcal{A} . In the ideal world, \mathcal{S} has the ability to corrupt entities and inform $\mathcal{F}_{\text{SA}^2\text{FE}}$ accordingly. In the subsequent process, $\mathcal{F}_{\text{SA}^2\text{FE}}$ will discard all messages from the corrupted party and treat \mathcal{S} as the corrupted party.

Lemma 2. $\text{Exec}_{\text{Game1}, \mathcal{Z}} \approx \text{Exec}_{\text{Game2}, \mathcal{Z}}$.

Proof. Simulator \mathcal{S} obtains setup information from SP and FA. Upon receiving registration requests from SP, ES, and user, \mathcal{S} has sufficient information to generate messages acceptable to $\mathcal{F}_{\text{register}}$. This allows $\mathcal{F}_{\text{register}}$ to update the internal tables T_s , T_p , T_t accordingly. Specifically, at ES registers to BS stage, \mathcal{S} records the map between the puzzle from real world and the $\text{puzzle}_{\text{ideal}}$ in the ideal world. At token registration stage, \mathcal{S} maintains a local map \mathcal{R} that associates token with $\text{token}_{\text{ideal}}$.

For offloading and payment claim, considering the threat model in Section III-B, the following cases need to be tackled.

1) *Wrong token from user side:* Consider a user who tries to double spend a token or use a forged token to get the service from multiple ESs. If the user generates a fake token that deceives the BS, the unforgeability of the blind signature is violated. In the real world, the BS validates user's token and rejects the request for invalid or double-spent tokens. In the ideal world, \mathcal{S} checks and updates \mathcal{R} to reject tokens where f_{bs} is not fresh. \mathcal{S} creates a nonexistent $\text{token}_{\text{ideal}}$ in T_t for an invalid token, causing $\mathcal{F}_{\text{offload}}$ to return "invalidToken" back.

2) *User exploits a specific ES:* A user linking two re-randomized puzzles will lead to a violation of the DBDH/DDH assumptions. The user can only exploit a specific ES by reusing the same z_u as before. If BS detects that z_u has been used before, it rejects the offloading request. In the ideal world, \mathcal{S} retrieves $\text{puzzle}'_{\text{ideal}}$ of z_u and sends it to $\mathcal{F}_{\text{offload}}$. If $\mathcal{F}_{\text{offload}}$ finds that $\text{puzzle}'_{\text{ideal}}$ for uid is not with ver^* or $\text{puzzle}'_{\text{ideal}}$ in T_p has $f_p = \text{used}$, $\mathcal{F}_{\text{offload}}$ returns an "invalidPuzzle" message to \mathcal{S} . \mathcal{S} then responds to the user with "fail".

3) *BS/ES exaggerates service for rewards:* If the BS/ES uses an invalid token or double spends it to claim extra rewards, the FA will detect it and reject the request. In the ideal world, if \mathcal{S} generates a $\text{token}_{\text{ideal}}$ not existing in T_t for an invalid token or the f_{bs}/f_{es} of $\text{token}_{\text{ideal}}$ is claimed, $\mathcal{F}_{\text{claim}}$ will return an "invalidToken" message to \mathcal{S} .

4) *Curious BS on user service request:* For puzzle registration from an ES, upon receiving $(\text{puzzle}, \text{ID}_{BS})$, \mathcal{S} forwards it to the BS. The rerandomized puzzle generated by the BS requires no translation by \mathcal{S} . For the offloading request, selected puzzle and (z_u, ct) from the user, \mathcal{S} directly forwards the user's output to the BS. As \mathcal{S} only performs forwarding, the real and ideal worlds are indistinguishable.

5) *Curious FA, SP, BS, ES on user identity:* The blind signature proof process resembles the ticket request process in [34], which has been proved to be UC-secure. \square

Game 3: In this game, the simulator \mathcal{S} cannot directly communicate with honest parties. Instead, \mathcal{S} needs to generate the outputs of the honest parties to the adversary \mathcal{A} .

Lemma 3. $\text{Exec}_{\text{Game2}, \mathcal{Z}} \approx \text{Exec}_{\text{Game3}, \mathcal{Z}}$.

Proof. \mathcal{S} generates the system parameters and keys on behalf of the honest parties. In the ideal world, upon receiving a registration request from the corrupted/honest ES or user, \mathcal{S} constructs a corresponding message and sends it to $\mathcal{F}_{\text{register}}$. \mathcal{S} generates real-world puzzle , token , and other information using these parameters and keys, and then sends them to \mathcal{A} .

For offloading requests from corrupted users, \mathcal{S} rerandomizes the puzzle and records its mapping to $\text{puzzle}_{\text{new}}$. This enables \mathcal{S} to construct real-world puzzle lists indistinguishable within and across batches by using universal re-encryption or bilinear mapping to generate real-world puzzles matching ideal-world random strings, which it then sends to \mathcal{A} .

For the payment claim protocol, when a corrupted BS/ES claims the reward, \mathcal{S} validates the token and generates corresponding $\text{token}_{\text{ideal}}$, and sends it to $\mathcal{F}_{\text{claim}}$. After receiving feedback, \mathcal{S} generates corresponding messages for \mathcal{A} . \square

Combining Lemmas 1–3 proves Theorem 2. \square

TABLE II
COMMUNICATION COST AND EXECUTION TIME OF SA²FE

Description	Communication Cost*		Execution Time	
	Message	Size (bytes)	Step	Time (ms)
Setup	-	-	SP setup	371.30
			FA setup	392.42
SP registration	SP to FA registration request	2012	SP registration	59.74
ES registration	ES to SP registration response	748	ES registered to SP	2.62
	ES to BS registration request	275	ES puzzle generation time	3.12
			ES registered to BS	1.57
User registration	User to FA registration request	1317	User blinded token	1.71
	User to FA registration response	1626	FA signed token	18.65
			User got token	58.89
			User unblinded token	0.84
			User verified token	1.73
Offloading	User initial offloading request	1310	BS rerandomized puzzles	55.92
	User received puzzle list	8163	User got response of initial offloading request	62.19
	User generated service request	309	User selected puzzle	237.67
			User got service response	6.88
Payment claim	BS token claim request	1310	BS claimed token	2.24
	ES token claim request	1312	ES claimed token	2.67

* Sizes of trivial text messages such as “success” or “fail” are omitted. The message size excludes the service content data ciphertext.

TABLE III
EVALUATION PLATFORMS

Platform	CPU	OS	Memory
HWI-AL00 Phone	Hisilicon Kirin 960 2.36GHz, 8 cores	Android 8.0.0 (ARM)	6GB
Raspberry Pi 4 Model B	Broadcom BCM2835 700MHz, 4 cores	Ubuntu 22.10 (ARM)	3.7GB
Laptop	Intel Core i7-7700HQ 2.80GHz, 8 cores	64-bit Windows (x86)	24GB
Desktop	AMD Ryzen 3945WX 4.0GHz, 12 cores	64-bit Ubuntu (x86)	256GB

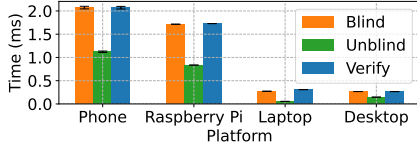


Fig. 6. Token registration (blind signature) delay.

VII. PERFORMANCE EVALUATION

A. Implementation and Experiment Settings

We used the gRPC framework (v.1.51.3) [44] to implement the communication between different parties. All protocols were implemented in Python. We used RSA blind signature for blindly signing the tokens, and used AES in CBC mode for symmetric key encryption, both implemented in the Crypto library (v.3.17) [45]. For the puzzle based on bilinear map, we used the pairing library from Charm Crypto (v.0.50) [46] and used the SS512 curve on x86 platform, and the PBC library (v0.5.14) (in C language) on ARM CPU platform. The puzzle based on universal re-encryption was implemented based on the ElGamal encryption scheme [47]. SHA-256 was used as the hash function in the protocol.

We evaluated the performance of SA²FE on four platforms as shown in Table III. By default, the user client was run on the Raspberry Pi, while the other parties were run on the desktop.

B. Evaluation Results

We evaluated the performance of SA²FE by analyzing both the communication overhead and execution time. Table II presents the evaluation results, showing the added overhead of our solution compared to ordinary offloading, where service requests are directly offloaded from the user to a edge server without any security guarantees. We conducted a statistical

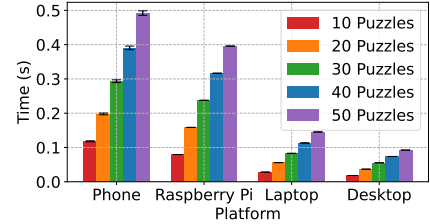


Fig. 7. Bilinear map puzzle matching delay.

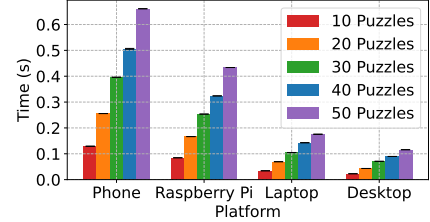


Fig. 8. Universal re-encryption puzzle matching delay.

analysis on 1,000 runs of SA²FE on different platforms to calculate the average time taken for each step. The default number of puzzles was set to 30.

As shown in the communication cost part of Table II, all message sizes were below 9KB. The practical execution time part of Table II provides an overview of the delay associated with each step. SP setup and FA setup phases had the longest delays, which should only be executed once when the system initializes. The puzzle generation overhead for ES was relatively small, with merely a 3.12ms overhead. At the user end, the most significant delay during the offloading phase occurred when selecting a puzzle. It took approximately 237.67ms to match and randomly choose one from 30 available puzzles. This selection process occurs only once at offloading initiation, while the duration of a single service session can last for a considerable time, such as minutes to hours [48], [49].

To evaluate the overhead on the user side of SA²FE, we collected the computation delays on user registration and puzzle matching processes on four different platforms. The results are presented in Figs. 6–8, with error bars representing the 95% confidence interval obtained from running each experiment 1000 times. Fig. 6 shows the delay experienced by the user during the registration. It can be observed that the user-side

computation overhead during the user registration process was small. Even on the lowest-performing platform, the average delay for each step was at most 2.07ms. Figs. 7 and 8 focus on two different puzzle implementation approaches: bilinear map and universal re-encryption. These figures illustrate the match delay for various numbers of puzzles on different platforms. It can be observed that with better computation capability, the match delay decreased. Additionally, as the number of puzzles increased, the delay in selecting all the suitable puzzles from the puzzle list and randomly choosing one puzzle as the final puzzle also increased. Overall, while DBDH is a stronger assumption compared to DDH, the bilinear map puzzle exhibited slightly lower computation overhead compared to the universal re-encryption puzzle with the current implementation.

VIII. CONCLUSION

In this paper, we proposed SA²FE, an anonymous, auditable and fair service offloading framework for democratized edge computing ecosystem. A novel rerandomizable puzzle primitive was introduced to enhance the design of the service offloading by preserving service type privacy and enabling fair and randomized edge server selection. Additionally, a token-based scheme was proposed to enable access control, maintain user anonymity, protect service type confidentiality, and enable accountable token verification and claiming. We proved the security of SA²FE based on the UC framework. The experimental results demonstrated the efficiency of SA²FE in terms of communication and computation overhead.

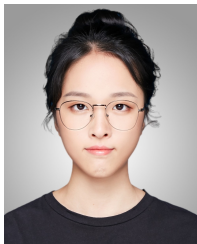
REFERENCES

- [1] H. Duan, J. Li, S. Fan, Z. Lin, X. Wu, and W. Cai, "Metaverse for social good: A university campus prototype," in *ACM MM*, 2021, pp. 153–161.
- [2] Z. Meng, T. Wang, Y. Shen, B. Wang, M. Xu, R. Han, H. Liu, V. Arun, H. Hu, and X. Wei, "Enabling high quality real-time communications with adaptive frame-rate," in *USENIX NSDI*, 2023, pp. 1429–1450.
- [3] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Karianakis, K. Hsieh, P. Bahl, and I. Stoica, "Ekya: Continuous learning of video analytics models on edge compute servers," in *USENIX NSDI*, 2022, pp. 119–135.
- [4] "NVIDIA unveils GPU-accelerated AI-on-5G system for edge AI, 5G and omniverse digital twins," accessed 2024-01-19. [Online]. Available: <https://blogs.nvidia.com/blog/2023/02/27/mwc-ai-on-5g-system/>
- [5] "100 edge computing companies to watch in 2023," accessed 2024-01-19. [Online]. Available: <https://stlpartners.com/articles/edge-computing/edge-computing-companies-2023/>
- [6] Z. Ning, P. Dong, X. Wang, S. Wang, X. Hu, S. Guo, T. Qiu, B. Hu, and R. Y. Kwok, "Distributed and dynamic service placement in pervasive edge computing networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 6, pp. 1277–1292, 2020.
- [7] R. Tourani, S. Srikanteswara, S. Misra, R. Chow, L. Yang, X. Liu, and Y. Zhang, "Democratizing the edge: A pervasive edge computing framework," *arXiv preprint arXiv:2007.00641*, 2020.
- [8] S. Dougherty, R. Tourani, G. Panwar, R. Vishwanathan, S. Misra, and S. Srikanteswara, "Apecs: A distributed access control framework for pervasive edge computing services," in *ACM CCS*, 2021, pp. 1405–1420.
- [9] D. Kaiser and M. Waldvogel, "Efficient privacy preserving multicast dns service discovery," in *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS)*. IEEE, 2014, pp. 1229–1236.
- [10] D. J. Wu, A. Taly, A. Shankar, and D. Boneh, "Privacy, discovery, and authentication for the internet of things," in *Computer Security—ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II 21*. Springer, 2016, pp. 301–319.
- [11] P. Welke, I. Andone, K. Blaszkiewicz, and A. Markowetz, "Differentiating smartphone users by app usage," in *ACM UbiComp*, 2016, pp. 519–523.
- [12] M. Weiss, M. Luck, R. Girgis, C. Pal, and J. P. Cohen, "A survey of mobile computing for the visually impaired," *arXiv preprint arXiv:1811.10120*, 2018.
- [13] F. Zhu, M. Mutka, and L. Ni, "Prudentexposure: A private and user-centric service discovery protocol," in *Second IEEE Annual Conference on Pervasive Computing and Communications, 2004. Proceedings of the*. IEEE, 2004, pp. 329–338.
- [14] X. Zhou, D. He, J. Ning, M. Luo, and X. Huang, "Aadec: Anonymous and auditable distributed access control for edge computing services," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 290–303, 2022.
- [15] K. Xue, W. Chen, W. Li, J. Hong, and P. Hong, "Combining data owner-side and cloud-side access control for encrypted cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 2062–2074, 2018.
- [16] T. Zheng, Y. Luo, T. Zhou, and Z. Cai, "Towards differential access control and privacy-preserving for secure media data sharing in the cloud," *Computers & Security*, vol. 113, p. 102553, 2022.
- [17] Y. Hu, S. Kumar, and R. A. Popa, "Ghostor: Toward a secure data-sharing system from decentralized trust," in *USENIX NSDI*, 2020, pp. 851–877.
- [18] H. Li, J. Yu, J. Fan, and Y. Pi, "Dsos: A distributed secure outsourcing system for edge computing service in iot," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 53, no. 1, pp. 238–250, 2022.
- [19] Y. Mao, W. Hong, H. Wang, Q. Li, and S. Zhong, "Privacy-preserving computation offloading for parallel deep neural networks training," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1777–1788, 2020.
- [20] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, "New algorithms for secure outsourcing of modular exponentiations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 9, pp. 2386–2396, 2013.
- [21] X. Chen, Y. Cai, Q. Shi, M. Zhao, B. Champagne, and L. Hanzo, "Efficient resource allocation for relay-assisted computation offloading in mobile-edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 2452–2468, 2019.
- [22] M. Gao, R. Shen, L. Shi, W. Qi, J. Li, and Y. Li, "Task partitioning and offloading in dnn-task enabled mobile edge computing networks," *IEEE Transactions on Mobile Computing*, 2021.
- [23] Y. Gao, W. Tang, M. Wu, P. Yang, and L. Dan, "Dynamic social-aware computation offloading for low-latency communications in iot," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7864–7877, 2019.
- [24] G. S. Park and H. Song, "Cooperative base station caching and x2 link traffic offloading system for video streaming over sdn-enabled 5g networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 2005–2019, 2018.
- [25] L. Zhang, M. Wang, L. Wang, Z. Chen, and H. Zhang, "Optimizing vehicle edge computing task offloading at intersections: a fuzzy decision-making approach," *The Journal of Supercomputing*, vol. 81, no. 1, p. 29, 2025.
- [26] M. Jia, L. Zhang, J. Wu, Q. Guo, G. Zhang, and X. Gu, "Deep multi-agent reinforcement learning for task offloading and resource allocation in satellite edge computing," *IEEE Internet of Things Journal*, 2024.
- [27] Y. Chen, J. Zhao, Y. Wu, J. Huang, and X. Shen, "Multi-user task offloading in uav-assisted leo satellite edge computing: A game-theoretic approach," *IEEE Transactions on Mobile Computing*, 2024.
- [28] S. Wang, Z. Lu, H. Gui, X. He, S. Zhao, Z. Fan, Y. Zhang, and S. Pang, "Ddqn-based online computation offloading and application caching for dynamic edge computing service management," *Ad Hoc Networks*, p. 103681, 2024.
- [29] H. Sun, Y. Zhou, H. Zhang, L. Ale, H. Dai, and N. Zhang, "Joint optimization of caching, computing and trajectory planning in aerial mobile edge computing networks: A maddpg approach," *IEEE Internet of Things Journal*, 2024.
- [30] C. Li, X. Deng, R. Huang, L. Zheng, and C. Yang, "Edge computing offload and resource allocation strategy with pairing theory," in *International Conference on Mobile Multimedia Communications*. Springer, 2025, pp. 283–295.
- [31] R. Aloufi, H. Haddadi, and D. Boyle, "Edgy: On-device paralinguistic privacy protection," in *ACM MobiCom Workshop*, 2021, pp. 3–5.
- [32] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Y. Zomaya, and B. B. Zhou, "Profiling applications for virtual machine placement in clouds," in *IEEE CLOUD*, 2011, pp. 660–667.
- [33] G. Fuchsbauer, A. Plouviez, and Y. Seurin, "Blind schnorr signatures and signed elgamal encryption in the algebraic group model," in *IACR EUROCRYPT*, 2020, pp. 63–95.

- [34] M. S. Turan, "Tmps: Ticket-mediated password strengthening," in *CT-RSA*, vol. 12006, 2020, p. 225.
- [35] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.
- [36] P. Golle, M. Jakobsson, A. Juels, and P. Syverson, "Universal re-encryption for mixnets," in *CT-RSA*, 2004, pp. 163–178.
- [37] M. Green and G. Ateniese, "Identity-based proxy re-encryption," in *ACNS*, 2007, pp. 288–306.
- [38] Y. Tsiounis and M. Yung, "On the security of elgamal based encryption," in *IACR PKC*, 1998, pp. 117–134.
- [39] X. Wang, R. Yu, D. Yang, H. Gu, and Z. Li, "Veriedge: Verifying and enforcing service level agreements for pervasive edge computing," in *IEEE INFOCOM*, 2024, pp. 2149–2158.
- [40] A. Kate, E. V. Mangipudi, S. Maradana, and P. Mukherjee, "Flexirand: Output private (distributed) vrf's and application to blockchains," in *ACM CCS*, 2023, pp. 1776–1790.
- [41] X. Hao, C. Lin, W. Dong, X. Huang, and H. Xiong, "Robust and secure federated learning against hybrid attacks: A generic architecture," *IEEE Transactions on Information Forensics and Security*, 2023.
- [42] M. Ben-Or, M. Horodecki, D. W. Leung, D. Mayers, and J. Oppenheim, "The universal composable security of quantum key distribution," in *IACR TCC*, 2005, pp. 386–406.
- [43] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *IEEE FOCS*, 2001, pp. 136–145.
- [44] "gRPC: A high performance, open-source universal RPC framework," accessed 2024-01-19. [Online]. Available: <https://grpc.io/>
- [45] "PyCrypto: The Python cryptography toolkit," accessed 2024-01-19. [Online]. Available: <https://github.com/pycrypto/pycrypto>
- [46] "Charm: A framework for rapidly prototyping cryptosystems," accessed 2024-01-19. [Online]. Available: <https://github.com/JHUISI/charm>
- [47] "Python implementation of the elgamal crypto system," accessed 2024-01-19. [Online]. Available: <https://github.com/RyanRiddle/elgamal>
- [48] V. Farhadi, F. Mehmeti, T. He, T. F. La Porta, H. Khamfroush, S. Wang, K. S. Chan, and K. Poularakis, "Service placement and request scheduling for data-intensive applications in edge clouds," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 779–792, 2021.
- [49] A. Aral and I. Brandic, "Dependency mining for service resilience at the edge," in *IEEE/ACM SEC*, 2018, pp. 228–242.



Xiaojian Wang (Student Member 2021) received her B.E. degree from Taiyuan University of Technology, China, in 2017 and received her M.S. degree in Computer Science from University of West Florida, FL, USA and Taiyuan University of Technology, China, in 2020. She is now a Ph.D. student in the department of Computer Science, College of Engineering at North Carolina State University. Her research interests include satellite network, security, blockchain, edge computing and so on.



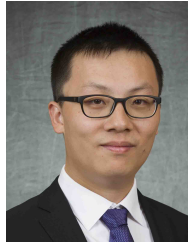
Huayue Gu (Student Member 2021) received her B.E. degree (2019) in Computer Science from Nanjing University of Posts and Telecommunications, Jiangsu, China, and M.S. degree (2021) in Computer Science from University of California, Riverside, CA, USA. Currently, she is a Ph.D. student in the Computer Science department at North Carolina State University. Her research interests are quantum networking, quantum communication, data analytics, etc.



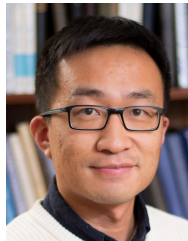
Zhouyu Li (Student Member 2021) received his B.S. degree from Central South University, Changsha, China, in 2019 and his M.S. degree from Georgia Institute of Technology, Atlanta, U.S., in 2020. Currently, he is a Ph.D. student of Computer Science at North Carolina State University. His research interests include privacy, cloud/edge computing, network routing, etc.



Fangtong Zhou (Student Member 2021) received her B.E. degree (2018) in Electrical Engineering and Automation from Harbin Institute of Technology, Harbin, China and M.S. degree (2020) in Electrical Engineering from Texas A&M University, College Station, Texas, USA. Currently she is a Ph.D. candidate in the School of Computer Science at North Carolina State University. Her research interests include machine learning in computer networking, like federated learning, reinforcement learning for resource provisioning, etc.



Ruozhou Yu (Student Member 2013, Member 2019, Senior Member 2021) is an Assistant Professor of Computer Science at North Carolina State University, USA. He received his PhD degree (2019) in Computer Science from Arizona State University, USA. His research interests include internet-of-things, cloud/edge computing, smart networking, algorithms and optimization, distributed machine learning, security and privacy, blockchain, and quantum networking. He has served or is serving on the organizing committees of IEEE INFOCOM 2022-2023 and IEEE IPCCC 2020-2023, as a TPC Track Chair for IEEE ICCCN 2023, and as members of the technical committee of IEEE INFOCOM 2020-2024 and ACM Mobihoc 2023. He is a recipient of the NSF CAREER Award in 2021.



Dejun Yang (Senior Member, IEEE) received the B.S. degree in computer science from Peking University, Beijing, China, and the Ph.D. degree in computer science from Arizona State University, Tempe, AZ, USA. He is currently an Associate Professor of Computer Science with the Colorado School of Mines, Golden, CO, USA. His research interests include the Internet of Things, networking, and mobile sensing and computing with a focus on the application of game theory, optimization, algorithm design, and machine learning to resource allocation, security, and privacy problems. Prof. Yang has received the IEEE Communications Society William R. Bennett Prize in 2019. He has served as the TPC Vice-Chair for Information Systems for IEEE International Conference on Computer Communications (INFOCOM). He currently serves an Associate Editor for the IEEE Transactions on Mobile Computing, IEEE Transactions on Network Science and Engineering, and IEEE Internet of Things Journal.



Guoliang Xue (Member 1996, Senior Member 1999, Fellow 2011) is a Professor of Computer Science in the School of Computing and Augmented Intelligence at Arizona State University. His research interests span the areas of Internet-of-things, cloud/edge/quantum computing and networking, crowdsourcing and truth discovery, QoS provisioning and network optimization, security and privacy, optimization and machine learning. He received the IEEE Communications Society William R. Bennett Prize in 2019. He is an Associate Editor of IEEE Transactions on Mobile Computing, as well as a member of the Steering Committee of this journal. He served on the editorial boards of IEEE/ACM Transactions on Networking and IEEE Network Magazine, as well as the Area Editor of IEEE Transactions on Wireless Communications, overseeing 13 editors in the Wireless Networking area. He has served as VP-Conferences of the IEEE Communications Society. He is the Steering Committee Chair of IEEE INFOCOM.