This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3478049

ProMark: Ensuring Transparency and Privacy-Awareness in Proximity Marketing Advertising Campaigns

Anh-Tu Hoang, Barbara Carminati *Senior Member,* and Elena Ferrari *IEEE Fellow*

***Abstract*—Advertising campaigns are crucial in business development, but most marketing techniques target online purchases (e.g., Google Adsense) and rely on a centralized architecture to store and process the campaign's data and check its effectiveness. Recently, proximity marketing has become more popular thanks to the widespread use of smartphones. It exploits the short-range communication (e.g., Bluetooth) between smartphones and beacon devices to collect and send marketing information to customers. However, this might create privacy issues for customers due to the potential leakage of sensitive information (such as locations associated with time). In this paper, we propose ProMark, a privacy-aware blockchain-based platform to verify the effectiveness of proximity marketing campaigns by ensuring transparency, decentralization, and privacy in the measurement process. We implemented ProMark and carried out experiments that show that ProMark can be used in super-regional malls even during peak hours.**

***Index Terms*—proximity marketing, privacy, transparency, blockchain.**

I. Introduction

Marketing plays a crucial role in business growth by delivering product information to customers. This is achieved through online marketing, utilizing online channels (e.g., emails, social networks), and offline channels (e.g., newspapers, television), to attract customers to online/offline stores. Despite extensive research on the impact of marketing campaigns on online purchases, their effects on offline stores like retail stores have received less attention.

Major advertisers are now using proximity technologies, similar to those in contact-tracing apps[1], to run and assess marketing campaigns in retail stores. These technologies utilize short-range communication technologies (e.g., Bluetooth Low Energy, WiFi, NFC) to interact with nearby customers' smartphones. For instance, companies like Purple[2] and Beaconstac[3] have developed tools to monitor customer behaviors in retail stores, measuring campaign effectiveness based on visitor numbers. Amazon utilizes this data to improve customers' shopping experiences in Amazon Go stores by suggesting relevant products. However, these solutions operate in a centralized manner, lacking transparency and compromising customer privacy in the process of verifying campaign effectiveness. This creates opportunities for dishonest advertisers to exploit marketing platform attacks (e.g., hit inflation, hit shaving) and raises concerns about privacy due to extensive data collection.

Several methods [1], [2], [3] aim to protect customer privacy in marketing platforms. [1] allows users to privately search for relevant nearby products without disclosing their queries while [2] recommends products to users while preserving the privacy of users' behaviour data. [3] gives users more control on their data while interacting with proximity devices. While claiming to protect customer privacy by not storing explicit identifiers such as names or emails, there is no evidence to suggest that these approaches prevents malicious advertisers from re-identifying customers by tracking their data over time. Furthermore, these methods do not address the evaluation of advertising campaign effectiveness.

Other proposals leverage blockchain to ensure transparency. TAVLA [4] utilize blockchain to enhance transparency in vehicular marketing networks, with the lack of transparently measuring campaign effectiveness. Basic Attention Token (BAT)[5] offers a blockchain-based digital marketing system transparently assessing online campaign effectiveness while protecting privacy. However, BAT cannot evaluate campaigns promoting offline stores as it relies on the Brave browser[4], which does not capture offline behaviors. In contrast, we are not aware of proposals addressing the transparency of measuring advertising campaigns' effectiveness for retail stores while protecting customers' privacy.

To address these issues, we propose *ProMark* – Proximity and Privacy-Aware Marketing Platform, built on Hyperledger Fabric[5]. It enables collaboration between advertisers, who promote their products/stores, and publishers, who distribute the products/stores' information to customers, in transparent and privacy-preserving proximity marketing campaigns. Smart contracts encode all activities, ensuring adherence to marketing procedures. Moreover, we introduce the Proof-of-Campaign (PoC), a non-interactive zero-knowledge proof protocol, allowing advertisers and publishers to evaluate campaign performance without compromising privacy and security. PoC uses Pedersen Commitment [6], the Elliptic Curve Integrated Encryption Scheme [7], and multi-party computation techniques to generate tokens proving customer participation in

Anh-Tu Hoang (tu.hoang@tuhh.de) is with the Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things, Institute of Data Engineering, Hamburg University of Technology, Germany,

Barbara Carminati (barbara.carminati@uninsubria.it) and Elena Ferrari (elena.ferrari@uninsubria.it) are with the Department of Theoretical and Applied Science (DiSTA), University of Insubria, Italy.

[1]https://www.google.com/covid19/exposurenotifications

[2]https://purple.ai/products/marketing-automation/proximity-marketing

[3]https://www.beaconstac.com

[4]https://brave.com

[5]https://www.hyperledger.org/use/fabric

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3478049

marketing campaigns. *ProMark* addresses security and privacy issues: hit inflation, crowd fraud, hit shaving, unauthorized effectiveness measurement, and customers' privacy. The experimental performance results indicate its feasibility for use in super-regional malls.

The paper is structured as follows. Section II discusses related work. Section III provides necessary background knowledge. Section IV presents a general proximity marketing workflow. Section V offers *ProMark*'s overview and the considered threat model. Section VI describes implementation details. Section VII proves privacy/security properties, while Section VIII shows performance experiments. Finally, Section IX concludes the paper. The supplementary contains time complexity analyses, proofs of privacy/security properties, the notations' summary, and additional experimental results.

II. Related Work

We are aware of few work addressing the same use case as *ProMark*. In this section, we discusses privacy-aware proximity marketing approaches and blockchain-based marketing platforms.

Privacy-aware Proximity Marketing. Liu et al. [1] use Elgamal [7] and Pedersen Commitment [6] to build a private set intersection protocol matching customer queries and advertisers' product descriptions without leaking query keywords. Rykowski et al. [2] develop a proximity marketing platform collecting customer behaviors in offline stores and recommending relevant products based on this data. Chahal et al. [3] enable users to opt out of marketing campaigns and prevent beacon devices from collecting data from them. Their extension [8] safeguards users' privacy by randomly swapping user devices' MAC address with those of other users.

Blockchain-based Marketing. TAVLA [4] indexes keywords from product descriptions in blockchain, allowing customers to query nearby products. As users' queries are not stored, the campaigns' effectiveness cannot be measured. Far et al. [9] designed a blockchain-based multi-level marketing framework allowing publishers to earn revenues from the sales of their subordinate publishers while safeguarding the anonymity of the latter. However, this work only considers the publishers' privacy. BAT [5] utilizes Brave Browser to collect online campaign participation data while safeguarding customer privacy. It is unsuitable for offline campaigns as it cannot collect participation data in offline stores.

In contrast, *ProMark* stores only zero-knowledge proofs of customer participation in blockchain. The proofs prevent users' sensitive information from exposure while allowing authorized advertisers and publishers to utilize them to assess the effectiveness of their marketing campaigns. By using multi-party computation techniques, attackers must compromise all campaign publishers' and advertisers' verifiers to breach customer privacy.

III. Background

In this section, we first present Hyperledger Fabric. Then, we describe the Pedersen Commitment Scheme used in the proposed zero-knowledge proof protocol.

A. Hyperledger Fabric

Blockchain [10] is an immutable transaction ledger maintained within a peer-to-peer network. Transactions are results of smart contracts, self-executed programs, that needs to be validated by all peers before adding to the ledger. There are two types: permissionless (e.g., Ethereum, Bitcoin) and permissioned (e.g., Hyperledger Fabric). Permissionless blockchains allow anyone to join and verify transactions, relying on economic incentives for protection. However, they may not suit enterprise needs due to fee instability and data privacy concerns. Permissioned blockchains, maintained by identified participants, are better suited for enterprise applications. Hence, we use Hyperledger Fabric, a leading permissioned blockchain, in this paper.

Hyperledger Fabric lets organizations join a blockchain network, managing peers and certificate authorities (CAs). Peers execute smart contracts upon requests from authenticated members. The network relies on endorsement policies to validate transactions, with the default policy requiring endorsement from at least one peer per organization.

Hyperledger Fabric allows organizations to create channels to restrict access to transactions, inviting other organizations to join. Each channel has its endorsement policies, and only participating organizations' peers are informed of its transactions. Organizations can establish private data collections within the channel, accessible solely by a subset of organizations. Unauthorized organizations' peers only receive hashes of sensitive data for validation.

B. Pedersen Commitment Scheme

The Pedersen Commitment Scheme [6] allows a party to commit to a value without revealing it. This commitment can be shared with others to prove the possession of the value while keeping it hidden. We adopt elliptic curves variations of this scheme as in Monero [11] and Zether [12] for stronger protection. Initially, all parties agree on parameters: i) an elliptic curve E defined over a finite field $\mathbb{F}_p$: $E(\mathbb{F}_p)$, where p is a large prime number; ii) a random generator $G \in E(\mathbb{F}_p)$; iii) a selected $H \in E(\mathbb{F}_p)$ such that finding $x_H \in \mathbb{Z}_p$ satisfying $H = x_H G$ is computationally infeasible. To commit to a secret integer s, a party generates a random integer $r \in \mathbb{Z}_p$ and forms the commitment as follows:

$$Comm(s, r) = sG + rH \tag{1}$$

The commitment has additive homomorphic property. For every s, r, s_1, s_2, r_1, $r_2 \in \mathbb{Z}_p$ such that if $s = s_1 + s_2$ and $r = r_1 + r_2$, the following equation holds:

$$Comm(s, r) = Comm(s_1, r_1) + Comm(s_2, r_2) \tag{2}$$

IV. Proximity Marketing

This section describes the main actors and the underlying process involved in a proximity advertising campaign. The notations adopted in the paper are presented in Supplementary I.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3478049

A. Main Actors

Advertisers: These are companies that want to promote their products/stores to customers.

Publishers: A publisher deploys an advertising campaign distributing the advertiser's products/stores' information to customers via various marketing channels and is compensated based on campaign effectiveness.

Customers: Customers, who have the ProMark smartphone application installed, receive information about promoted products/stores through various marketing channels like news, social networks, or beacon devices.

Witness devices: A witness device, often a beacon, utilizes short-range communication technologies such as NFC or Bluetooth to interact with smartphones through protocols like iBeacon[6] and Eddystone[7]. Placed at store entrances or near advertised products, these devices interact with customers' proximity marketing apps. Their goal is to collect proofs that nearby customers received campaign information, encoded as a *token*. If a customer interacts with the campaign notification, this token is downloaded to their smartphone.

After agreeing to deploy their proximity marketing campaign, advertisers and publishers proceed with the process outlined in the following section.

B. Proximity Marketing Process

The process starts with an advertiser a contacting a publisher p to deploy a marketing campaign c whose types are: *store attraction* or *product attraction*. Store attraction campaigns promote sets of stores (e.g., newly opened ones), while product attraction campaigns focus on specific products. We formally define a marketing campaign as follows:

Definition 1 (Marketing campaign) A marketing campaign c is modelled as a tuple $\langle c_{id}, name, description, a_{id}, p_{id}, t_s, t_e, stores, product, min^c, \mathcal{W}^c \rangle$, where c_{id}, $name$, $description$ are c's unique ID, name, and description; a_{id}, p_{id} are the unique IDs of c's advertiser and publisher; t_s, t_e are c's start and end time; $stores$ is the set of stores affected by the campaign; $product$ is c's promoted product; min^c is the minimum number of customers to which p must distribute c's information; and $\mathcal{W}^c$ lists authorized witness devices IDs. If $product$ is empty, c is a store attraction campaign. Otherwise, it is a product attraction campaign.

Once an advertiser and a publisher agree on a campaign c, the publisher advertises the product/stores to selected customers.[8] If the customer reacts to c's information (e.g., click on c's notification), a *token* is downloaded to his/her smartphone. Tokens are then used to evaluate c's effectiveness. For this purpose, the publisher installs witness devices in stores or product areas affected by c. For store attraction campaigns, devices are deployed at store entrances, while for product-targeted campaigns, devices are placed where the promoted product is located. When customers pass by the witness devices, they connect with the publisher's app to verify if their smartphones contain c's token. The presence of the token confirms that customers received c's information and came to c's promoted store/product, indicating c is effectiveness. More tokens collected imply a more effective campaign.

Once campaign c ends, its publisher/advertiser can evaluate c's performance based on collected tokens. To measure the effectiveness of a campaign, we use two popular metrics: *foot traffic* and *conversion rate* [13].

A campaign c's foot traffic ($FT(c)$) represents the total visits to the promoted stores during the campaign duration. A higher $FT(c)$ indicates greater c's effectiveness.[9]

Although foot traffic's increases shows campaign effectiveness, it does not confirm customer visits are prompted by receiving advertising information. When multiple campaigns run simultaneously for the same store or product, foot traffic alone cannot identify which campaigns are most effective. For this purpose, we introduce the *offline conversion rate*, an extension of the conversion rate metric[13], to calculate the percentage of customers who visited and stayed at the promoted product/store's area for a time interval (e.g., 15 minutes) after obtaining the campaign tokens, over those who received the tokens.

Definition 2 (Offline Conversion Rate) The offline conversion rate (OCR) of a campaign c is the ratio of the number of times customers who hold c's tokens spent a specific time interval Δ_{stay} in the area of c's promoted product/stores to the number of c's tokens sent to all customers:

$$OCR(c) = \frac{n^t_{ocr}(c)}{n_{ocr}(c)}$$

where $n_{ocr}(c)$ is the number of c's tokens sent to customers and $n^t_{ocr}(c)$ is the number of times customers who hold c's tokens stay at least a period Δ_{stay} in the area of c's promoted product/stores between c's start and end time.

Proximity marketing has benefits to advertisers (e.g., collecting/analyzing customers' traffic and interests) and customers (e.g., receiving/participating in marketing campaigns according to customers' locations), but it requires trust among the involved parties. Advertisers must trust publishers in accurately measuring campaign performance, while customers need to trust that their personal information (e.g., name, arrival time, location) remains secure. To address these concerns, we present *ProMark* enabling advertisers to validate campaign performance measurement while safeguarding customer data.

V. ProMark

This section describes the overview of *ProMark* and the related potential privacy/security issues.

A. Overview

ProMark utilizes Hyperledger Fabric to facilitate transparent interaction among all major actors. *ProMark* blockchain comprises peers belonging to advertisers and publishers. Fig. 1 depicts *ProMark*'s workflow. Every step of the proximity

[6]https://developer.apple.com/ibeacon/

[7]https://github.com/google/eddystone

[8]Publisher's selection of customers based on their location or profile in proximity marketing is beyond the scope of this paper.

[9]Note that if the same customer visits the store multiple times, these visits are all counted in $FT(c)$.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3478049

marketing process is executed through smart contracts, as detailed in Section VI.

The process starts with an advertiser deploying a campaign c via Smart Contract *SC_Initialization* (step 1), which records c's parameters in a transaction (cfr. Definition 1). Witness devices associated with c (i.e., $\mathcal{W}^c$) must obtain *witness tokens* through Smart Contract *SC_Witness_Token_Generation* (step 2) to authorize themselves for collecting c's tokens and adding collected tokens without disclosing their identities.

Outside the blockchain platform, the publisher starts advertising c's product/stores to selected customers. Once a customer responds to the campaign notification, the *ProMark* application calls Smart Contract *SC_Campaign_Token_Generation* (step 3) to generate a *campaign token*, stored in the customer's smartphone.

When the customer passes nearby the promoted stores/product, authorized witness devices communicate with the *ProMark* application to collect the *campaign token* from the customer's smartphone (step 4.A). For each collected campaign token, witness devices call Smart Contract *SC_Token_Collection* to create a transaction storing the collected token and their witness token (step 4.B).

Both advertisers and publishers can retrieve transactions created in c's lifetime by using Smart Contract *SC_Verification* (step 5). The retrieved transactions enable computation of Foot Traffic and Offline Conversion Rate (cfr. Section IV).

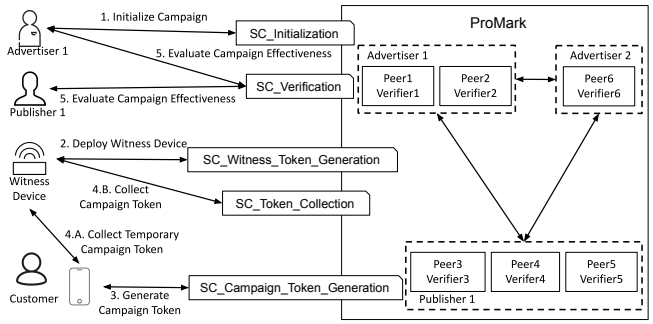


Fig. 1. ProMark workflow.

B. Threat Model

We categorize *ProMark* participants into internal and external groups, aligning with common practices in industrial proximity-aware platforms utilizing IoT devices (e.g., FOAM[10], Amazon Go, Beaconstac). Internal participants, authorized to access *ProMark* directly, include advertiser/publisher peers, witness devices, and the *ProMark* application. *ProMark* relies on Hyperledger Fabric to authorize and manage internal participants' access. Thus, we consider them as semi-honest (aka honest-but-curious), adhering to *ProMark*'s workflow (outlined in Section V-A), but may seek to extract sensitive information from exchanged messages. External participants (i.e., advertisers, publishers, and customers) join *ProMark* through internal participants. Following the state-of-the-art [14], we assume external participants may be malicious, capable of conducting various attacks on the system to gain benefits.

In what follows, we analyze how external participants exploit internal ones to perform attacks on *ProMark*. We exclude discussions on blockchain platform vulnerabilities, like 51% attacks, Sybil attacks, and block withholding, as these are platform-specific and less risky in Hyperledger Fabric [15].

Issue 1-Hit Inflation: A dishonest publisher p might try to increase the performance of its deployed campaign c to be paid more by exploiting witness devices (e.g., to increase the number of collected tokens).

Issue 2-Crowd Fraud: A dishonest publisher p could employ dishonest customers to visit the advertised store entrance/promoted product area of a campaign c to boost its performance. This action aims to artificially inflate c's performance by increasing the number of interactions with c's witness devices.

Issue 3-Hit Shaving: A dishonest advertiser a tries to lower c's performance by ignoring a portion of the collected tokens so that a pays its hired publisher p less money for c.

Issue 4-Unauthorized Effectiveness Measurement: Transactions about collected tokens contain information associating them with their campaign c (e.g., the witness device's id adding tokens, and its publisher, and c's id). Dishonest advertisers and/or publishers, via their peers, could monitor tokens of other advertisers/publishers' campaigns and calculate their effectiveness.

It is important to note that, in Hyperledger Fabric, the access to campaigns' transactions can be restricted to their advertisers/publishers by using channel/private data collection. Thus, an advertiser and a publisher agree on a campaign, they can create a *channel/private data collection* to store the campaign's transactions whose parameters are inaccessible to other advertisers and publishers.

However, the restriction does not prevent attackers from gaining malicious access to publishers' peers. Then, the attackers could read transactions regarding collected tokens and use them to evaluate the effectiveness of the campaign.

Issue 5-Customers' Privacy: A dishonest advertiser or publisher's peer of a campaign c could monitor transactions to link a customer's identifier to their temporal locations.

Section VII explains how *ProMark* addresses all the above mentioned issues.

VI. ProMark Implementation

This section details *ProMark*'s implementation focusing on verifying: (1) whether a collected token belongs to a given campaign, and (2) if the token is collected by an authorized device of the campaign. To this end, we design two token types: the *campaign token* for customers and the *witness token* for devices. The former confirms customers' receipts of campaigns' promoted information, while the later shows devices' rights to collect campaigns' tokens.

To prevent dishonest advertisers/publishers from exploiting the tokens to compromise customer privacy or manipulate campaign effectiveness, we introduce the *Proof-of-Campaign* (PoC) protocol. PoC is a non-interactive zero-knowledge proof

[10]https://www.foam.space/

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3478049

protocol with two key features: 1) both the advertiser and publisher must collaborate in the token generation and verification steps, ensuring mutual agreement and preventing unilateral modifications to campaign effectiveness and 2) transactions containing collected tokens must not store identifiers of customers and devices collected them.

Moreover, to prevent Hit Inflation (Issue 1) and Crowd Fraud (Issue 2), *ProMark* uses two parameters: Δ_{stay} (the minimum customer stay time within device range), and Δ_{token} (the validity interval of a token). Witness devices use Δ_{stay} to verify customer interest in products/stores, while Δ_{token} limits token addition frequency to one per Δ_{token} interval, as explained in Section VII.

In the following, we first present the PoC protocol (Section VI-A) and detail *ProMark*'s steps (Sections VI-B-VI-F).

A. Proof-of-Campaign

PoC is a protocol for generating and verifying *campaign tokens* and *witness tokens*. Throughout the paper, we refer to entities storing tokens as clients including application installed in customer smartphones and witness device. PoC generates and verifies tokens via multi-party computation techniques, requiring collaboration between publishers and advertisers using their secret values. The client maintains the token's secrecy and derives one-time tokens (also known as temporary tokens) from it to ensure non-traceability during verification. The collaboration between the advertiser and publisher is necessary to verify the one-time tokens. PoC verification suffices to link one-time versions of a token to its campaign. We formally define PoC as follows:

Definition 3 (Proof-of-Campaign) Let c be a campaign deployed by an advertiser a and a publisher p; $\mathcal{V}_a$, $\mathcal{V}_p$ be the set of verifiers of a and p, respectively; and $\mathcal{V}^c$ be the set of selected verifiers for campaign c, such that $\mathcal{V}^c$ includes at least one verifier from $\mathcal{V}_a$ and $\mathcal{V}_p$.

Setup: All verifiers in $\mathcal{V}^c$ agree on the Pedersen Commitment Scheme's parameters (cfr. Section III-B): $\mathbb{F}_p$, G, H. Each verifier $v_i \in \mathcal{V}^c$ holds a secret value s_i^c for generating/verifying all c's tokens.

Generation: A client cli requests a PoC token for campaign c from publisher p. Publisher p asks c's verifiers ($\mathcal{V}^c$) to generate the token. Each verifier $v_i \in \mathcal{V}^c$ generates a random integer $r^c_{cli,i}$, calculates the commitment $C^c_{cli,i} = Comm(s^c_i, r^c_{cli,i})$, and returns $C^c_{cli,i}, r^c_{cli,i}$ to the publisher. The publisher constructs the token: $token^c_{cli} = \langle C^c_{cli}, r^c_{cli}\rangle$, where $C^c_{cli} = \sum_i^{n^c} C^c_{cli,i}$ and $r^c_{cli} = \sum_i^{n^c} r^c_{cli,i}$, $n^c = |\mathcal{V}^c|$ and returns $token^c_{cli}$ to cli. cli keeps $token^c_{cli}$ private.

When client cli wants to prove possession of token $token^c_{cli}$, it creates the *temporary version* of the token at time t: $token^c_{cli,t} = \langle \overline{C}^c_{cli,t,1}, \ldots, \overline{C}^c_{cli,t,n^c}, \overline{r}^c_{cli,t,1}, \ldots, \overline{r}^c_{cli,t,n^c}\rangle$, where $\overline{C}^c_{cli,t,i} = Enc(P_i, C^c_{cli,t,i})$, $\overline{r}^c_{cli,t,i} = Enc(P_i, r^c_{cli,t,i})$, P_i is the public key of v_i, Enc is the encryption function, such that $C^c_{cli} = \sum_i^{n^c} C^c_{cli,t,i}$ and $r^c_{cli} = \sum_i^{n^c} r^c_{cli,t,i}$. $token^c_{cli,t}$ is valid within the period $[t, t + \Delta_{token})$, where Δ_{token} is the validity duration of all temporary tokens.

Verification: To verify if client cli holds a PoC token of campaign c, publisher p verifies the temporary token $token^c_{cli,t}$. p sends $\overline{C}^c_{cli,t,i}$ and $\overline{r}^c_{cli,t,i}$ to verifier $v_i \in \mathcal{V}^c$. Each verifier v_i decrypts them to obtain $C^c_{cli,t,i}$, $r^c_{cli,t,i}$; calculates $C'^c_{cli,t,i} = Comm(s^c_i, r^c_{cli,t,i})$; and sends $C^c_{cli,t,i}$, $C'^c_{cli,t,i}$ back to p for verification. If $\sum_i^{n^c} C^c_{cli,t,i} = \sum_i^{n^c} C'^c_{cli,t,i}$, $token^c_{cli,t}$ is valid; otherwise, it is invalid.

When a campaign is added to *ProMark*, its verifiers execute the PoC setup step. The generation and verification steps of PoC are managed by functions *Generate_Token()*, *Generate_Temporary_Token()*, and *Verify_Token()*, respectively.

Function *Generate_Token()* (Function 1) takes as input the id of a campaign c: c_{id}; the id of a client cli: cli_{id}; and the set of verifiers of c: $\mathcal{V}^c$. First, it initializes the commitment C^c_{cli} and r^c_{cli} with the identity element O of Pedersen Commitment's elliptic curve and 0 (line 1- 2). The identity element is a point in the curve such that all points in the curve are unchanged after adding it [7]. For every verifier $v_i \in \mathcal{V}^c$, it requests v_i to call Function *Calculate_Commitment()* that generates a single random integer $r^c_{cli,i}$ and commitment $C^c_{cli,i}$ for client cli, regardless of how many times it is called for the same campaign c and client cli (line 4). The function updates C^c_{cli} and r^c_{cli} by using $C^c_{cli,i}$ and $r^c_{cli,i}$ (lines 5-6). Finally, it constructs and returns $token^c_{cli} = \langle C^c_{cli}, r^c_{cli}\rangle$ (lines 8-9).

Function *Calculate_Commitment()* (Function 2) returns only one version of $C^c_{cli,i}$ and $r^c_{cli,i}$ for the pair of campaign c and client cli, preventing Hit Inflation (Issue 1) and Crowd Fraud (Issue 2). Each verifier v_i maintains a Redis database

Function 1 Generate_Token($c_{id}, cli_{id}, \mathcal{V}^c$)

Input: c_{id}: the id of the campaign c; cli_{id}: the id of a client cli; $\mathcal{V}^c$: the set of verifiers of campaign c.
Output: $token^c_{cli}$: PoC token.

1: $C^c_{cli} \leftarrow O$
2: $r^c_{cli} \leftarrow 0$
3: **for** $v_i \in \mathcal{V}^c$ **do**
4: $\quad C^c_{cli,i}, r^c_{cli,i} \leftarrow v_i$'s Calculate_Commitment($c_{id}, cli_{id}$)
5: $\quad C^c_{cli} \leftarrow C^c_{cli} + C^c_{cli,i}$
6: $\quad r^c_{cli} \leftarrow r^c_{cli} + r^c_{cli,i}$
7: **end for**
8: $token^c_{cli} \leftarrow \langle C^c_{cli}, r^c_{cli}\rangle$
9: **return** $token^c_{cli}$

Function 2 Calculate_Commitment(c_{id}, cli_{id})

Input: c_{id}: the id of the campaign c; cli_{id}: the id of a client cli.
Output: $C^c_{cli,i}$: the commitment and $r^c_{cli,i}$: a random integer.

1: Watch($\langle c_{id}, cli_{id}\rangle$)
2: $r^c_{cli,i} \leftarrow$Read_By_Key(c_{id}, cli_{id})
3: **if** $r^c_{cli,i} = \varnothing$ **then**
4: $\quad r^c_{cli,i} \leftarrow$Generate_Random_Integer()
5: $\quad$ **if** Store($cli_{id}, r^c_{cli,i}$) = FALSE **then**
6: $\quad\quad r^c_{cli,i} \leftarrow$Read_By_Key($c_{id}, cli_{id}$)
7: $\quad$ **end if**
8: **end if**
9: $s^c_i \leftarrow$Read_By_Key(c_{id})
10: $C^c_{cli,i} \leftarrow$Commit($s^c_i, r^c_{cli,i}$)
11: **return** $C^c_{cli,i}, r^c_{cli,i}$

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3478049

where it stores a key $\langle c_{id}, cli_{id}\rangle$ and the corresponding value $r^c_{cli,i}$. When multiple peers request v_i to execute this function simultaneously, Redis' optimistic locking mechanism[11] is used to prevent a key's associated value from being replaced.

This function first checks whether the value associated with the key $\langle c_{id}, cli_{id}\rangle$ has changed by calling Function *Watch()* (line 1). This ensures that concurrent executions of *Calculate_Commitment()* cannot store another version of $r^c_{cli,i}$ with the same key. Then, it finds the integer $r^c_{cli,i}$ associated with the key $\langle c_{id}, cli_{id}\rangle$ in its Redis database (line 2). If the key does not exist, it generates a random integer $r^c_{cli,i}$ and stores it along with the key using Function *Store()* (lines 4-5). If Store() fails due to a concurrent addition, the function reads the stored integer $r^c_{cli,i}$ from the database (line 6). Finally, it reads s^c_i associated with the key c_{id} from the database, calculates the commitment $C^c_{cli,i}$ (Equation 1), and returns $C^c_{cli,i}$ and $r^c_{cli,i}$ (lines 9- 11).

Function 3 Generate_Temporary_Token($token^c_{cli}$, Δ_{token}, n^c)

Input: $token^c_{cli}$: the PoC token of a client cli; Δ_{token}: the valid duration of temporary token; n^c: the number of campaign c's verifiers.

Output: $token^c_{cli,t}$: the temporary version of $token^c_{cli}$ at time t.

1: Let t' be the timestamp when the previous temporary version of $token^c_{cli}$ has been generated.
2: **if** $t - t' < \Delta_{token}$ **then**
3: $\quad token^c_{cli,t} \leftarrow token^c_{cli,t'}$
4: **else**
5: $\quad \sum_c \leftarrow O$
6: $\quad \sum_r \leftarrow O$
7: $\quad$ **for** $i := 0; i < n^c; i++$ **do**
8: $\quad\quad$ **if** $i < n^c - 1$ **then**
9: $\quad\quad\quad$ Let $C^c_{cli,t,i}$ be a random point in $E(\mathbb{F}_p)$.
10: $\quad\quad\quad$ Let $r^c_{cli,t,i}$ be a random integer in $\mathbb{F}_p$.
11: $\quad\quad\quad \sum_c \leftarrow \sum_c + C^c_{cli,t,i}$
12: $\quad\quad\quad \sum_r \leftarrow \sum_r + r^c_{cli,t,i}$
13: $\quad\quad$ **else**
14: $\quad\quad\quad C^c_{cli,t,i} \leftarrow C^c_{cli} - \sum_c$
15: $\quad\quad\quad r^c_{cli,t,i} \leftarrow r^c_{cli} - \sum_r$
16: $\quad\quad$ **end if**
17: $\quad\quad \overline{C}^c_{cli,t,i} \leftarrow$Enc$(P_i, C^c_{cli,t,i})$
18: $\quad\quad \overline{r}^c_{cli,t,i} \leftarrow$Enc$(P_i, r^c_{cli,t,i})$
19: $\quad$ **end for**
20: $\quad token^c_{cli,t} = \langle \overline{C}^c_{cli,t,1}, \ldots, \overline{C}^c_{cli,t,n^c}, \overline{r}^c_{cli,t,1}, \ldots, \overline{r}^c_{cli,t,n^c}\rangle$
21: **end if**
22: **return** $token^c_{cli,t}$

Function *Generate_Temporary_Token()* (Function 3) takes a token $token^c_{cli}$, the temporary token's valid duration Δ_{token}, and the number of c's verifiers n^c as inputs. It constructs the temporary version of $token^c_{cli}$ at time t: $token^c_{cli,t}$. First, it gets the timestamp t' of the previous temporary token generation (line 1). If the previous temporary token was generated later than Δ_{token}, it assigns $token^c_{cli,t}$ as the previous token $token^c_{cli,t'}$ (line 3). This ensures uniqueness of $token^c_{cli,t}$ in the ledger, limiting client cli to adding only one token every Δ_{token} and preventing Hit Inflation (Issue 1) and Crowd Fraud (Issue 2). If not, a new temporary token is generated (lines 4-21). The function initializes the sums of all commitments $\sum_c$ and random integers $\sum_r$ as the identity element O and 0, respectively (lines 5-6). For $i < n^c - 1$, it initializes random point $C^c_{cli,t,i} \in E(\mathbb{F}_p)$ and integer $r^c_{cli,t,i} \in \mathbb{Z}_p$, then adds them to $\sum_c$ and $\sum_r$ (lines 9-12). The last commitment and integer are computed by subtracting $\sum_c$ and $\sum_r$ from C^c_{cli} and r^c_{cli}, respectively (lines 14-15). The function encrypts $C^c_{cli,t,i}$ and $r^c_{cli,t,i}$ with v_i's public key as $\overline{C}^c_{cli,t,i}$ and $\overline{r}^c_{cli,t,i}$, resp.(lines 17-18). Finally, it constructs and returns $token^c_{cli,t}$ (line 20-22).

Function 4 Verify_Token(c_{id}, $token^c_{cli,t}$, $\mathcal{V}^c$)

Input: c_{id}: the id of a campaign c; $token^c_{cli,t}$: the temporary token; $\mathcal{V}^c$: the set of campaign c's verifiers.

Output: $\mathcal{I}^c_t$: a Boolean value indicating the validity of $token^c_{cli,t}$.

1: $\sum'_c \leftarrow O$
2: $\sum_c \leftarrow O$
3: **for** $v_i \in \mathcal{V}^c$ **do**
4: $\quad C'^c_{cli,t,i}, C^c_{cli,t,i} \leftarrow$call v_i's Verify_Commitment($\overline{r}^c_{cli,t,i}$, $\overline{C}^c_{cli,t,i}$)
5: $\quad \sum'_c \leftarrow \sum'_c + C'^c_{cli,t,i}$
6: $\quad \sum_c \leftarrow \sum_c + C^c_{cli,t,i}$
7: **end for**
8: **if** $\sum'_c = \sum_c$ **then**
9: $\quad \mathcal{I}^c_t \leftarrow TRUE$
10: **else**
11: $\quad \mathcal{I}^c_t \leftarrow FALSE$
12: **end if**
13: **return** $\mathcal{I}^c_t$

Function *Verify_Token()* (Function 4) validates the association between a temporary token $token^c_{cli,t}$ and a campaign c's id: c_{id} according to Sc's verifiers: $\mathcal{V}^c$. First, it initializes the sums of commitments $\sum'_c$ and points $\sum_c$ as the identity element O (lines 1-2). For every verifier $v_i \in \mathcal{V}^c$, it requests v_i to decrypt and calculate the commitment for verification (line 4). v_i decrypts $\overline{r}^c_{cli,t,i}$, $\overline{C}^c_{cli,t,i}$ to obtain $r^c_{cli,t,i}$, $C^c_{cli,t,i}$, computes $C'^c_{cli,t,i} = Comm(s^c_i, r^c_{cli,t,i})$, and returns $C'^c_{cli,t,i}$ and $C^c_{cli,t,i}$. These values update $\sum'_c$ and $\sum_c$ (lines 5-6) If $\sum_c$ matches $\sum'_c$, $\mathcal{I}^c_t$ is set to $TRUE$ (line 9), indicating the token's validity; otherwise, it is set to $FALSE$ (line 11). Finally, it returns $\mathcal{I}^c_t$ (line 13).

We modify Hyperledger Fabric's peers to serve both as regular peers, participating in distributed consensus for endorsing smart contracts' results, and as verifiers, responsible for generating and validating campaign and witness tokens according to the protocol.

PoC requires trusted execution of functions *Generate_Token()* and *Verify_Token()*, but advertisers and publishers may be malicious in our threat model (cfr. Section V-B). *ProMark* solves this by executing the functions in smart contracts (i.e., *SC_Campaign_Token_Generation*, *SC_Witness_Token_Generation*, *SC_Verification*). This ensures correct token generation without mutual trust among partic-

[11] https://redis.io/docs/manual/transactions/\#optimistic-locking-using-check-and-set

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3478049

Smart Contract 1 SC_Initialization($Param^c, \mathcal{V}^c$)

Input: $Param^c$: parameters of campaign c; $\mathcal{V}^c$: the set of c's verifiers.

1: Let $Cert_a$ be the digital certificate of the advertiser who calls the smart contract.
2: Let $c_{id}, a_{id}, p_{id}, t_s, t_e$ be the id of c, advertiser, publisher, start time, and end time in $Param^c$.
3: Let $\mathcal{A}$ and $\mathcal{P}$ be the set of advertisers and publishers.
4: Let $\mathcal{V}_a, \mathcal{V}_p$ be the set of verifiers of a and p.
5: **if** (c_{id} does not exist) $\wedge$ ($a_{id} \in \mathcal{A}$) $\wedge$ ($p_{id} \in \mathcal{P}$) $\wedge$ ($t_s < t_e$) $\wedge$ ($Cert_a$ is the digital certificate of advertiser a) $\wedge$ ($\mathcal{V}^c \subseteq \mathcal{V}_a \cup \mathcal{V}_p$) $\wedge$ ($\mathcal{V}^c \cap \mathcal{V}_a \neq \emptyset$) $\wedge$ ($\mathcal{V}^c \cap \mathcal{V}_p \neq \emptyset$) **then**
6: **for** $v_i \in \mathcal{V}^c$ **do**
7: Call v_i's Initialize_Secret_Value(c_{id})
8: **end for**
9: $CT^c \leftarrow \langle Param^c, \mathcal{V}^c \rangle$
10: Store CT^c into the ledger.
11: **end if**

ipants. Fig. 1 depicts how PoC protocol is integrated into *ProMark*'s workflow, which we detail in following sections.

B. Campaign Initialization

This step allows an advertiser a to create a campaign transaction for its campaign c through Smart Contract *SC_Initialization* (cfr. Smart Contract 1).

A campaign transaction includes the campaign parameters (cf. Definition 1) and the agreed set of verifiers, determined by the advertiser and publisher before executing *SC_Initialization*. It is represented as $CT^c = \langle Param^c, \mathcal{V}^c \rangle$, where $Param^c$ denotes campaign parameters and $\mathcal{V}^c$ represents c's verifiers.

SC_Initialization takes as input $Param^c$ and campaign c's verifiers: $\mathcal{V}^c$. It begins by obtaining the advertiser's digital certificate: $Cert_a$, followed by extracting c_{id}, a_{id}, p_{id}, t_s, and t_e from $Param^c$ (line 1-2). Then, it fetches the set of all advertisers and publishers in *ProMark*'s network: $\mathcal{A}$, $\mathcal{P}$ along with their respective verifiers: $\mathcal{V}_a$, $\mathcal{V}_p$ (line 3-4). It verifies c's validity, ensuring that (1) c's id is unique; (2) the advertiser and publisher are recognized; (3) $t_s < t_e$; (4) $Cert_a$ corresponds to a, (5) $\mathcal{V}^c$ belongs to $\mathcal{V}_a$ and $\mathcal{V}_p$; and (6) $\mathcal{V}^c$ includes at least one verifier from both a and p. IF all conditions are met, it prompts each verifier $v_i \in \mathcal{V}^c$ to initialize a secret value s_i^c for c (line 7). This initialization guarantees that each verifier generates only one s_i^c for c, irrespective of the number of initializations. Finally, it creates and stores CT^c to the ledger, requiring majority approval from organizations (line 9-10).

C. Witness Device Deployment

After the parameters of a campaign c are added to the ledger, all of its witness devices (i.e., $\mathcal{W}^c$) receive c_{id} and execute the smart contract *SC_Witness_Token_Generation* to obtain their witness tokens (cfr. Smart contract 2).

SC_Witness_Token_Generation takes as input c_{id}: the id of campaign c. First, it obtains the certificate of the caller

Smart Contract 2 SC_Witness_Token_Generation(c_{id})

Input: c_{id}: id of a campaign c.
Output: $token_w^c$: the witness token of device w.

1: Let $Cert_w$ be the digital certificate of device w which calls the smart contract.
2: $\mathcal{W}^c \leftarrow$Get_Devices_By_Campaign(c_{id})
3: $token_w^c \leftarrow \emptyset$
4: **if** $Cert_w \in \mathcal{W}^c$ **then**
5: Let w_{id} be the id of the device w.
6: $\mathcal{V}^c \leftarrow$Get_Verifiers_By_Campaign(c_{id})
7: $token_w^c \leftarrow$Generate_Token($c_{id}, w_{id}, \mathcal{V}^c$)
8: **end if**
9: **return** $token_w^c$

device w: $Cert_w$ (line 1) and the set of c's witness devices: $\mathcal{W}^c$ (line 2). The witness token of w ($token_w^c$) is initialized as empty (line 3). Then, if $Cert_w$ belongs to a device in $\mathcal{W}^c$, it gets the id of the device w: w_{id} (lines 5). It calls Function *Get_Verifiers_By_Campaign()* with c_{id} to obtain c's verifiers: $\mathcal{V}^c$ (line 6). c_{id}, w_{id}, and $\mathcal{V}^c$ are passed to Function *Generate_Token()* (see Function 1 in Section VI-A) to generate w's witness token: $token_w^c$ (line 7). Finally, the smart contract returns $token_w^c$ to device w (line 9). $token_w^c$ must be endorsed by majority of peers in *ProMark* network.

D. Campaign Token Generation

When a customer u receives campaign c's information from c's publisher p, he/she uses the *ProMark* application to obtain a token for the campaign c. The application can obtain the id of the campaign c (c_{id}) from the received information (e.g., scanning its QR code, inputting the *id* of c, or receiving directly from beacon devices). Then, the application obtains a campaign token $token_u^c$ by calling the smart contract *SC_Campaign_Token_Generation*. The implementation of this smart contract is similar to that of *SC_Witness_Token_Generation*. Therefore, due to the lack of space, we do not present it in this paper.

E. Campaign Token Collection

In this step, *ProMark* utilizes witness devices to gather campaign tokens from participating customers.

When a customer u enters the range of a campaign c's device w at time t, their *ProMark* application generates the temporary version ($token_{u,t}^c$) of their campaign token $token_u^c$ using *Generate_Temporary_Token()* (Function 3). If the previous temporary token is invalid at time t, i.e., $t - t' \geq \Delta_{token}$, where t' is the timestamp of the previous temporary token, a new temporary token is generated. If customer u has not received the campaign token, $token_{u,t}^c$ remains empty. Subsequently, u's application forwards $token_{u,t}^c$ to device w. Upon u leaving w's range, the connection between w and u's *ProMark* is terminated. The device w uses Function *Generate_Temporary_Token()* to generate its temporary witness token and calls *SC_Token_Collection* (Smart Contract 3) to

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3478049

Smart Contract 3 SC_Token_Collection(c_{id}, $token^c_{u,t}$, $token^c_{w,t}$, Δ_t, t)

Input: c_{id}: the id of a campaign c; $token^c_{u,t}$: the temporary token of the campaign c for customer u; $token^c_{w,t}$: the temporary witness token of a device w; Δ_t: the time window customer u stays in the area of device w; t: the time the customer u arrived at the area of device w.

1: Let $Cert_w$ be the digital certificate of the device w that called the smart contract.
2: $\mathcal{W}^c \leftarrow Get\_Devices\_By\_Campaign(c_{id})$
3: **if** $Cert_w \in \mathcal{W}^c \wedge Verify\_Token(c_{id}, token^c_{w,t}) = TRUE \wedge \Delta_t \geq \Delta_{stay}$ **then**
4: $token_{id} \leftarrow Hash(token^c_{u,t})$
5: $TT^c_{u,t} \leftarrow Find\_Token\_By\_ID(token_{id})$
6: **if** $TT^c_{u,t} = \varnothing$ **then**
7: $TT^c_{u,t} \leftarrow \langle token^c_{u,t}, token^c_{w,t}, \Delta_t, t\rangle$
8: Store $TT^c_{u,t}$ in the ledger with id $token_{id}$.
9: **end if**
10: **end if**

create a token transaction storing the collected temporary campaign/witness token: $token^c_{u,t}$, $token^c_{w,t}$. A token transaction is formally defined as follows.

Definition 4 (Token Transaction) Let c be a marketing campaign, a be its advertiser, p be its publisher, w be a witness device and u be the customer whose c's token is collected by w at time t. A token transaction $TT^c_{u,t}$ is a tuple $\langle token^c_{u,t}, token^c_{w,t}, \Delta_t, t\rangle$, where $token^c_{u,t}$ is the campaign c's temporary token of customer u and $token^c_{w,t}$ is the temporary witness token of device w, Δ_t is the duration u stayed at the area of device w, and t is u's arrival time.

Smart Contract *SC_Token_Collection* (Smart Contract 3) allows a witness device to record its collected temporary campaign token in the ledger. It takes as input the campaign ID c_{id}, the collected temporary campaign token $token^c_{u,t}$, the temporary witness token $token^c_{w,t}$, the duration of customer u's stay Δ_t, and u's arrival time t. Here, the device obtained c_{id} and $token^c_{w,t}$ when it is deployed for campaign c (Section VI-C). It first gets the digital certificate of w which calls the smart contract: $Cert_w$ and c's devices $\mathcal{W}^c$ (line 1-2). If $\mathcal{W}^c$ includes $Cert_w$, $token^c_{w,t}$ was generated by c's verifiers, and $\Delta_t \geq \Delta_{stay}$, it proceeds to create the token transaction $TT^c_{u,t}$. It computes $TT^c_{u,t}$'s id: $token_{id}$ and finds transactions associated with $token_{id}$ ((line 4-5). If no transactions is found, it creates and stores $TT^c_{u,t}$ with id $token_{id}$ (line 7-8), requiring the majority approvals from *ProMark* network.

F. Campaign Effectiveness Evaluation

In this step, an advertiser a and a publisher p measure the effectiveness of their campaign c by using *offline conversion rate* (*OCR*) and *foot traffic* (*FT*) metrics (cfr. Section IV-B). Measuring *OCR* requires verifying transactions added during c's deployment period to ensure they originate from c's witness devices and contain tokens from customers who received c's campaign tokens. This verification is done by *SC_Verification* (Smart Contract 5).

Smart Contract 4 SC_Verification(c_{id})

Input: c_{id}: the id of a campaign c.
Output: n_{valid}: the number of transactions containing valid temporary tokens.

1: Let $Cert_m$ be the digital certificate of the advertiser/publisher who called the smart contract.
2: Let $t_s, t_e, \mathcal{V}^c, a, p$ be the start, end timestamp, and the set of verifiers, the advertiser, and the publisher of campaign c.
3: $n_{valid} \leftarrow 0$
4: **if** $Cert_m$is authorized by a and p **then**
5: $TTs \leftarrow Find\_Token\_Transactions(t_s, t_e)$
6: **for** $TT \in TTs$ **do**
7: Let $token^c_{w,t}$ and $token^c_{u,t}$ be the temporary device and campaign token in TT
8: **if** $Verify\_Token(c_{id}, token^c_{w,t}, \mathcal{V}^c) \wedge Verify\_Token(c_{id}, token^c_{u,t}, \mathcal{V}^c)$ **then**
9: $n_{valid} \leftarrow n_{valid} + 1$
10: **end if**
11: **end for**
12: **end if**
13: **return** n_{valid}

Function 5 Evaluate_Offline_Conversion_Rate(c_{id})

Input: c_{id}: the identity of a campaign c.
Output: ocr: the offline conversion rate of campaign c.

1: $n_{ocr} \leftarrow Count\_The\_Number\_Of\_Campaign\_Tokens(c_{id})$
2: $n^t_{ocr} \leftarrow SC\_Verification(c_{id})$
3: $ocr^c \leftarrow \frac{n^t_{ocr}}{n_{ocr}}$
4: **return** ocr^c

The smart contract takes a campaign c's id (c_{id}) as input and returns the count of valid transactions generated within c's start and end time. First, it gets the digital certificate of calling advertiser/publisher m and extracts c's parameters: t_s, t_e, $\mathcal{V}^c$, a, and p (line 1-2) If m is authorized by both the advertiser a and the publisher p, it finds the set of token transactions added within $[t_s, t_e]$: TTs (line 5). It initializes the number of valid transactions n_{valid} as 0 and starts verifying found transactions (lines 3-11). For each transaction TT in TTs, it verifies if temporary witness/campaign token $token^c_{w,t}$, $token^c_{u,t}$ are from c using *Verify_Token()* (Function 4) (line 8). If both tokens are valid, n_{valid} is increased by 1 (line 9). Finally, it returns n_{valid} (line 13).

Function 5 illustrates the use of *SC_Verification* to measure *OCR* (cfr. Definition 2). It is callable by advertisers and publishers. First, it finds the count of campaign tokens generated for campaign c: n_{ocr} (line 1). n_{ocr} can be obtained by accessing a verifier for c and tallying the random values used to generate its tokens. Next, it computes the number of customers arriving at c's witness devices' areas with c_{id} as input (n^t_{ocr}) through *SC_Verification* (line 2). Finally, it calculates *OCR* (ocr^c) by dividing n^t_{ocr} by n_{ocr} and returns it (lines 3-4).

To compute foot traffic (*FT*) as outlined in Section IV-B, we follow a similar procedure. However, since *FT* tracks customer

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3478049

visits to areas of campaign c's promoted products/stores during its deployment time, we count token transactions from c's witness devices during its duration. We can easily adapt *SC_Verification* to verify if a token transaction's temporary witness token originates from c's devices. Due to space limitations, we omit the detailed algorithm for measuring this metric and the modified *SC_Verification*.

Supplementary III describes the time complexity of the smart contracts and functions presented in this section.

VII. Privacy and Security Analysis

This section first states the completeness, soundness, and zero-knowledge properties of PoC (Theorems 1-3). Then, we prove how *ProMark* remedies issues 1-5 presented in Section V-B (cfr. Theorems 4-7). All the theorems follow the threat model described in Section V-B, while Supplementary II contains their formal proofs.

Theorem 1. (**Completeness**) Let c be a campaign and $\mathcal{V}^c$ be the set of c's verifiers. If customers/devices receive tokens for c and use them to generate temporary tokens, it is possible to verify that the temporary tokens belong to c.

The proof reported in Supplementary II shows that by holding a campaign's valid token $token^c_{cli}$, a client cli can generate a temporary token $token^c_{cli,t}$ such that the sums of $token^c_{cli,t}$'s commitments and random numbers are equal to $token^c_{cli}$'s commitment and random number. Therefore, $token^c_{cli,t}$ must be valid.

Theorem 2. (**Soundness**) Let c be a campaign and $\mathcal{V}^c$ be the set of c's verifiers. If customers/devices do not receive tokens from campaign c, it is impossible for them to generate temporary witness/campaign tokens belonging to c.

The proof reported in Supplementary II shows that if a client cli does not hold a valid token $token^c_{cli}$ of a campaign c and does not collude with c's verifiers, obtaining valid token is not possible. Even if cli generates a fake temporary token, the probability of its commitments and random numbers summing up to the valid token's commitment and random number is extremely low, akin to the probability of breaking asymmetric cryptographic schemes through brute force. Hence, generating a valid temporary token for c without possessing its valid token is infeasible.

Theorem 3. (**Zero-knowledge**) Verifiers of a campaign cannot exploit information collected from PoC to infer whether a temporary token belongs to a campaign.

The proof reported in Supplementary II starts from the observation that Verifiers, as semi-honest entities, operate independently without collusion. Through the PoC protocol, a verifier knows its secret s^c_i and individually monitor inputs/outputs of Function *Calculate_Commitment()* (i.e., c_{id}, cli_{id}, $C^c_{cli,i}$, $r^c_{cli,i}$) and Function *Verify_Commitment()* (i.e., $\overline{C}^c_{cli,t,i}$, $\overline{r}^c_{cli,t,i}$, $C'^c_{cli,t,i}$). Here, the verifier knows that $C^c_{cli,i}$ and $r^c_{cli,i}$ belong to c_{id}. Thus, to associate a temporary token (including $\overline{C}^c_{cli,t,i}$, $\overline{r}^c_{cli,t,i}$) with campaign c, the verifier must link it with $C^c_{cli,i}$ and $r^c_{cli,i}$. *ProMark* addresses this by randomly generating $\overline{C}^c_{cli,t,i}$, $\overline{r}^c_{cli,t,i}$ and allowing them to be added to the ledger only once, using *SC_Token_Collection()* and *Generate_Temporary_Token()*. Therefore, verifiers cannot exploit their collected information from PoC to infer the association between a temporary token and a campaign.

Theorem 4. (**Hit Inflation Protection**) Let p be a dishonest publisher, c be a proximity campaign, and u be a customer that passed by c's witness devices ($\mathcal{W}^c$). p cannot use witness devices of other campaigns to add u's temporary tokens for c and can only add at most one of u's tokens for c in the interval Δ_{token}.

A dishonest publisher p can execute the attack mentioned in the theorem above by relocating witness devices to boost the number of temporary tokens for campaign c First, p puts the devices authorized for other campaigns near c's promoted product/stores. Second, p positions c's devices in same areas allowing them to add a token many times. *ProMark* prevents these attacks with Smart Contract *SC_Token_Collection()*, ensuring that only c's devices can add their tokens, and a temporary token can only be added once.

Theorem 5. (**Crowd Fraud Protection**) Let u be a dishonest customer and c be the target campaign that u is hired to increase c's performance. u can add at most a temporary token for campaign c every Δ_{token}.

To execute crowd fraud attacks, a dishonest publisher p hires customers to pass by a campaign c's witness devices multiple times within a short period, boosting c's performance. To counter this, *ProMark* restricts each customer to generate at most one token transaction within the interval Δ_{token}, regardless of how many times they pass by the devices.

Hit Shaving: To achieve this, a dishonest advertiser a must ignore token transactions for its campaign c. This includes manipulating c's devices to ignore customers' temporary tokens and *SC_Token_Collection()* to not add transactions for c. However, given our assumptions that the devices follow *ProMark*'s procedure and Hyperledger Fabric ensures the consensus of *SC_Token_Collection()*'s implementation, advertiser a cannot ignore any token transactions to decrease c's performance.

Theorem 6. (**Unauthorized Effectiveness Measurement Protection**) Given a campaign c, if an attacker is not authorized by all of c's verifiers, they cannot verify whether c's temporary tokens belong to c.

Preventing this attack avoids that malicious users can measure the performance of a campaign they are not authorized for. To measure a campaign c's performance from other advertisers/publishers, a dishonest advertiser/publisher must retrieve and verify token transactions associated with c. To prevent this attack, *ProMark* restricts verification to authorized parties recognized by all of c's verifiers, including those from both c's publisher and advertiser. This ensures c's advertiser and publisher are notified when anyone verifies c's transactions. Moreover, attackers must compromise all of c's verifiers to infer c's performance.

Theorem 7. (**Customers' Privacy Protection**) An attacker cannot infer customers' temporal locations (i.e., their locations at specific times) by monitoring token transactions of a campaign c, if the attacker does not have access to the private keys of all verifiers in $\mathcal{V}^c$ and is not authorized by the verifiers.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3478049

Since c's devices in its promoted product/store areas, by knowing a customer u's token transactions, an attacker can infer u's temporal locations (u is at the promoted stores/product areas at specific times) if he/she knows u's token transactions. An attacker exploits u's participation in c at time t (i.e., t and Δ_t) to find the token transaction generated at that time. Then, he/she links this transaction with others stored in the ledger to deduce the rest of u's transactions, using u's temporary campaign token. *ProMark* addresses this through *SC_Token_Collection()* by ensuring that u's temporary campaign tokens are unique in all transactions. Therefore, even if the attacker finds one of u's transactions, he/she cannot link it to the rest of u's transactions.

VIII. Evaluation

In this section, we describe the simulation settings and experiments used to evaluate *ProMark*.

A. Simulation Setting

We deploy ProMark on Hyperledger Fabric 2.2, with smart contracts written in Go, whereas Encryption and Pedersen commitment are implemented using the Ristretto prime-order group over Edwards25519.[12] We use Docker to simulate different network architectures by varying parameters' organizations and peers per organization. Each organization endorses *ProMark* smart contract outputs through one peer, with performance unaffected by varying peer counts. Simulations run on an Ubuntu Server with a 32-core AMD Ryzen Threadripper 1950X CPU and 32GB RAM, utilizing Docker to execute Hyperledger Fabric's peers concurrently.

We evaluate smart contracts' performance using Hyperledger Caliper,[13] a tool for blockchain performance evaluation. It concurrently sends requests to smart contracts, measuring throughput (transactions per second) and latency (processing time). For read-only contracts (i.e., *SC_Verification*, *SC_Campaign_Token_Generation*, *SC_Witness_Token_Generation*), latency is the time from receiving inputs to validating results by the majority of network organizations. For writing contracts (i.e., *SC_Initialization*, *SC_Token_Collection*), latency is the time required to broadcast transactions to the majority of peers. We measure latency for all requests and calculate the average.

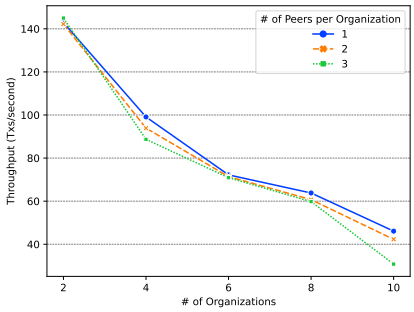
To gauge ProMark's feasibility in realistic settings, we compare smart contracts' throughput with the number of visitors to European super-regional malls. According to Statista, in 2023, European malls had an average monthly visitor count of $463,000$,[14] with the crowdest mall being Westfield Forum des Halles, which opens 71.5 hours per week. On peak month (i.e., December), the count increased by 14% .[15] We convert the monthly visitor count to 0.45 customers per second normally and 0.51 customers per second during the peak times. These values represent the expected throughput for *ProMark* in super-regional malls.

[12]https://github.com/bwesterb/go-ristretto

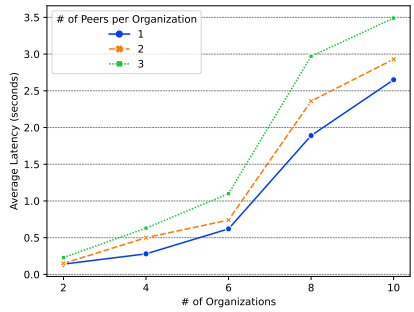
[13]https://hyperledger.github.io/caliper/

[14]https://www.statista.com/statistics/1445574/leading-shopping-centers-in-europe/

[15]https://storeforcesolutions.com/euro/retail-peak-hours-critical-in-driving-store-performance/



(a) Throughput



(b) Average Latency

Fig. 2. *SC_Initialization*'s performance by varying number of organizations and peers.

B. Experiments

We assess smart contract performance using four parameters: the number of: (a) organizations (n^o), (b) peers per organization (n^p), (c) verifiers per campaign (n^c), and (d) token transactions added during campaigns (n^t). n^o and n^p affect the time needed for peers to reach consensus of transactions while the time complexity of *ProMark* smart contracts relies on n^p, n^c, and n^t (see Supplementary III).

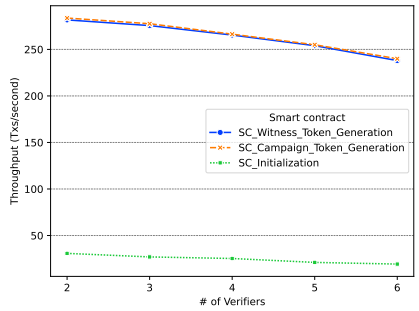
We evaluate performance by varying values of these parameters: n^o (2 to 10), n^p (1 to 3), n^c (2 to 6), and n^t ($115,750$; $231,500$; $463,000$). The network size ranges from 2 to 30 peers. The maximum n^c is twice n^p, and the total number of peers is among the highest reported in recent Hyperledger Fabric papers (e.g., 9 [16] or 20 peers[17]).

Due to space constraints, we first present performance results for *SC_Initialization* by varying n^o and n^p values (Fig. 2), since its time complexity is highest among all smart contracts (Supplementary III). Then, we evaluate *SC_Initialization*, *SC_Witness_Token_Generation*, and *SC_Campaign_Token_Generation* on the largest network (with $n^o = 10$ and $n^p = 3$), by varying n^c (Fig. 3). We also evaluate *SC_Verification* by varying n^c and n^t (Fig. 5). We assume marketing campaigns with three different durations: one week, two weeks, and four weeks, with added token transactions of $115,750$, $231,500$, and $463,000$, respectively (cfr. Section VIII-A).

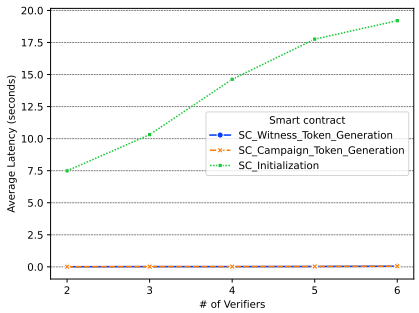
SC_Initialization. To assess the impact of n^o and n^p on this smart contract's performance, we measure its throughput

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3478049

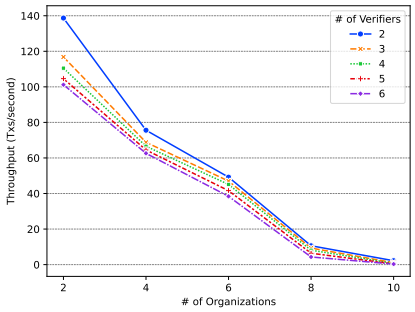


(a) Throughput

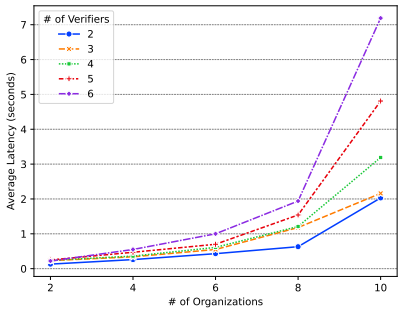


(b) Average Latency

Fig. 3. *SC_Witness_Token_Generation*, *SC_Campaign_Token_Generation*, and *SC_Initialization* performance by varying the number of verifiers.



(a) Throughput



(b) Average Latency

Fig. 4. *SC_Token_Collection*'s performance by varying the number of organizations and verifiers.

and latency when increasing n^p while fixing n^o and vice versa. Fig. 2 illustrates the performance. With $n^o = 10$, increasing n^p from 1 to 3 increases the total number of peers from 10 to 30. The increment decreases the throughput of the smart contract from 46.1 to 30.8 *tps* (transactions per second) while the average latency increases from 2.65 to 3.49 seconds. We simulate networks having similar number of peers per organization by fixing $n^p = 3$ and increasing n^o from 4 to 10. The throughput decreases from 88.7 to 30.8 *tps* while the average latency increases from 0.63 to 3.49 seconds. Increasing n^o and n^p decrease *SC_Initialization*'s performance but n^o has a higher impact than n^p. Since all *ProMark* smart contracts' outputs are endorsed by one peer of each organization, this trend is similar for all smart contracts. Fig. 3 illustrates the performance of *SC_Initialization* by varying n^c (i.e., $2, 3, 4, 5, 6$) in the largest network that has $n^o = 10$ and $n^p = 3$. Increasing n^c decreases *SC_Initialization*'s performance. Increasing n^c from 2 to 6 decreases the throughput from 30.8 to 19.3 *tps* while the average latency increases from 7.49 to 19.20 seconds. We recall that *SC_Initialization* is used to initialize parameters of marketing campaigns and the initialization does not need to be done in real-time. Therefore, the results indicate that this smart contract is feasible in practice.

SC_Witness_Token_Generation and *SC_Campaign_Token_Generation*. We recall that *SC_Witness_Token_Generation* is used to generate a device's witness tokens, whereas *SC_Campaign_Token_Generation* is used to generate customers' campaign tokens. Fig. 3 reports the performance by varying n^c. Since the performance of these smart contracts relies on the performance of Function *Generate_Token()* (Function 1), their experimental results are almost similar. Increasing n^c from 2 to 6 decreases the throughput of *SC_Witness_Token_Generation* from 281.7 to 238.0 *tps*, while that of *SC_Campaign_Token_Generation* decreases from 283.7 to 240.0 *tps*. *SC_Campaign_Token_Generation*'s throughput is much higher than the number of customers' visits in super-regional malls at peak hours (i.e., 0.51 customers per second). The average latency increases from 0.01 to 0.05 seconds. The latency is fast enough for the device/customer to receive the tokens instantaneously. Thus, these smart contracts are feasible to be used in super-regional malls even at peak hours.

SC_Token_Collection. Fig. 4 illustrates the performance of this smart contract by varying n^c and n^o, whereas $n^p = 3$. With $n^o = 2$, increasing n^c from 2 to 6 reduces the throughput from 138.7 to 101.2 *tps* with average latency rising from 0.13 to 0.22 seconds. Even with $n^o = 10$, the throughput ranges from 2.2 to 0.4 *tps* which is sufficient to support peak customer traffic in super-regional malls (i.e., 0.51 customers per second, see Section VIII-A). Thus, *SC_Token_Collection* efficiently collects campaign tokens even at peak hours.

SC_Verification. To assess this smart contract, we simulate the largest network of 30 peers ($n^o = 10$ and $n^p = 3$). To evaluate the performance in the worst case when all peers are in charge of verifying campaigns, we create campaigns such that all peers in the network must be verifiers of at least one campaign. Evaluation of any campaign necessitates *SC_Verification* to verify n^t token transactions. Due to space constraints, we only report the average latency of *SC_Verification* (Fig. 5) by varying n^c and n^t in a network

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3478049

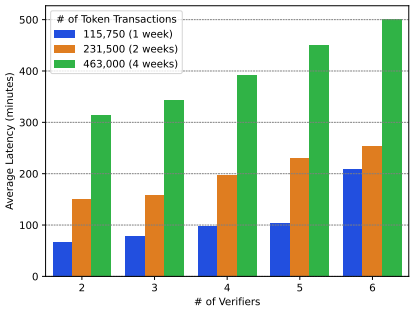


Fig. 5. *SC_Verification*'s performance on varying numbers of transactions.

with $n^o = 10$ and $n^p = 3$.

SC_Verification performance declines as n^c and n^t increase. With $n^t = 115,750$ (one-week campaigns), the average latency of *SC_Verification* rises from 66.39 to 207.98 minutes when n^c goes from 2 to 6. Similarly, with $n^t = 463,000$ (four-week campaigns), the average latency increases from 314.14 to 501.96 minutes. Theorem 6 demonstrated that the greater n^c is, the stronger the protection provided by *ProMark* against Unauthorized Effectiveness Measurement (Issue 4). To be noticed that this smart contract does not need to be executed in real-time. Instead, in practice, it is executed at the end of marketing campaigns. Therefore, even though verifying four-week campaigns with the highest protection ($n^c = 6$) requires 501.96 minutes to finish, it remains practical for use.

IX. Conclusion

This paper proposed *ProMark*, a blockchain based system to verify the effectiveness of proximity marketing campaigns, ensuring transparency of the process and privacy of the collected data. *ProMark*'s performance was evaluated by analyzing the time complexity of its smart contracts and conducting experiments, showing that it is applicable in super-regional malls even during peak hours. Even though multi-party computation techniques enhance *ProMark* security and privacy protection, it reduces *ProMark*'s performance when the number of verifiers increases. Leveraging zk-SNARK for generating proofs presents a promising avenue to tackle this challenge. This approach enables verifiable token generation and verification, ensuring customer privacy without necessitating collaboration from multiple verifiers. Moreover, we will extend *ProMark* to allow customers to be acknowledged for their campaigns' participation. The acknowledgment can result in transferable tokens, such as discount codes or cryptocurrency. Another direction for future work is to integrate *ProMark* with purchase platforms, such that *ProMark* can evaluate conversion rates according to customers' purchases.

References

[1] D. Liu *et al.*, "Efficient and privacy-preserving ad conversion for v2x-assisted proximity marketing," in *IEEE MASS*, 2018.
[2] J. Rykowski *et al.*, "In-store proximity marketing by means of iot devices," in *Collaborative Networks of Cognitive Systems*, 2018.
[3] F. Chahal *et al.*, "User control in proximity marketing: A basic consensus algorithm," in *IEEE Global Communications Conference*, 2023.
[4] D. Liu *et al.*, "Transparent and accountable vehicular local advertising with practical blockchain designs," *IEEE Transactions on Vehicular Technology*, pp. 15 694–15 705, 2020.
[5] B. Software, "Basic attention token (bat) blockchain based digital advertising (white paper)," Tech. Rep., 2021.
[6] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO'91*. Springer, 1991.
[7] D. Hankerson *et al.*, *Guide to Elliptic Curve Cryptography*. Springer, 2004.
[8] F. Chahal *et al.*, "Preserving user privacy and empowering control in proximity marketing," in *International Conference on Wireless Networks and Mobile Communications*, 2023, pp. 1–6.
[9] S. Banaeian Far *et al.*, "A privacy-preserving framework for blockchain-based multi-level marketing," *Computers & Industrial Engineering*, 2023.
[10] S. Shetty *et al.*, *Blockchain for Distributed Systems Security*. John Wiley & Sons, Ltd, 2019.
[11] K. M. Alonso *et al.*, "Monero - privacy in the blockchain," *IACR Cryptol. ePrint Arch.*, p. 535, 2018.
[12] B. Bünz *et al.*, "Zether: Towards privacy in a smart contract world," in *Financial Cryptography and Data Security*, 2020, pp. 423–443.
[13] J. A. Davis, *Measuring Marketing: The 100+ Essential Metrics Every Marketer Needs 3rd Edition*. Berlin, Boston: De Gruyter, 2018.
[14] Z. Pooranian *et al.*, "Online advertising security: Issues, taxonomy, and future directions," *CoRR*, vol. abs/2006.03986, 2020.
[15] S. Brotsis *et al.*, "On the security and privacy of hyperledger fabric: Challenges and open issues," in *IEEE SERVICES*, 2020.
[16] P. Ruan *et al.*, "Ledgerview: Access-control views on hyperledger fabric," in *ACM SIGMOD*, 2022, p. 2218–2231.
[17] T. Sato *et al.*, "Opssc: Decentralized blockchain network operation workflow for hyperledger fabric," in *IEEE Blockchain*, 2021.

Anh-Tu Hoang holds the position of Postdoctoral Researcher at the Hamburg University of Technology, Germany. He earned his Ph.D. in Computer Science from the University of Insubria, Italy. His research focuses on the development and privacy/security protection of decentralized systems, identity management, and machine learning.

Barbara Carminati is a professor of Computer science at the University of Insubria. She received her Ph.D. in Computer Science from the University of Milano, Italy. Her primary areas of interest in research are security and privacy for cutting-edge applications including cloud computing, semantic web, data outsourcing, XML data sources, Web services, and data streams, and online social networks. She has written more than 130 publications with peer reviews on these subjects. She is an IEEE and ACM senior member.

Elena Ferrari is a professor of Computer science at the University of Insubria, Italy, where she leads the STRICT Socialab. Her research activities are mainly related to data security, privacy and trust. Ferrari has a Ph.D. in Computer Science from the University of Milano, Italy. She received several awards, including the IEEE Computer Society's 2009 Technical achievement award for 'outstanding and innovative contributions to secure data management.' She is an IEEE and ACM fellow.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/