MRAAC: A Multi-stage Risk-aware Adaptive Authentication and Access Control Framework for Android

JIAYI CHEN, University of Waterloo, Waterloo, Canada
URS HENGARTNER, University of Waterloo, Waterloo, Canada
HASSAN KHAN, University of Guelph, Guelph, Canada

Adaptive authentication enables smartphones and enterprise apps to decide when and how to authenticate users based on contextual and behavioral factors. In practice, a system may employ multiple policies to adapt its authentication mechanisms and access controls to various scenarios. However, existing approaches suffer from contradictory or insecure adaptations, which may enable attackers to bypass the authentication system. Besides, most existing approaches are inflexible and do not provide desirable access controls. We design and build a multi-stage risk-aware adaptive authentication and access control framework (MRAAC), which provides the following novel contributions: **Multi-stage:** MRAAC organizes adaptation policies in multiple stages to handle different risk types and progressively adapts authentication mechanisms based on context, resource sensitivity, and user authenticity. **Appropriate access control:** MRAAC provides libraries to enable sensitive apps to manage the availability of their in-app resources based on MRAAC's risk awareness. **Extensible:** While existing proposals are tailored to cater to a single use case, MRAAC supports a variety of use cases with custom risk models. We exemplify these advantages of MRAAC by deploying it for three use cases: an enhanced version of Android Smart Lock, guest-aware continuous authentication, and corporate app for BYOD. We conduct experiments to quantify the CPU, memory, latency, and battery performance of MRAAC. Our evaluation shows that MRAAC enables various stakeholders (device manufacturers, enterprise and secure app developers) to provide complex adaptive authentication workflows on COTS Android with low processing and battery overhead.

CCS Concepts: • **Security and privacy** → **Multi-factor authentication**; **Mobile platform security;**

Additional Key Words and Phrases: Adaptive authentication, implicit authentication, behavioral biometrics, access control

ACM Reference Format:
Jiayi Chen, Urs Hengartner, and Hassan Khan. 2024. MRAAC: A Multi-stage Risk-aware Adaptive Authentication and Access Control Framework for Android. *ACM Trans. Priv. Sec.* 27, 2, Article 17 (April 2024), 30 pages. https://doi.org/10.1145/3648372

1 INTRODUCTION

Authentication systems are continuously evolving to provide more usable and secure options to the users. In addition to biometrics and multi-factor authentication, weak authenticators or

This research has been supported by the Waterloo-Huawei Joint Innovation Laboratory.
J. Chen present address: Huawei Technologies Canada Co., Ltd., 300 Hagey Blvd, Waterloo, N2L 0A4, ON, Canada.
Authors' addresses: J. Chen and U. Hengartner, University of Waterloo, 200 University Avenue West, Waterloo, ON, Canada, N2L 3G1; e-mails: jiayi.chen@uwaterloo.ca, urs.hengartner@uwaterloo.ca; H. Khan, University of Guelph, 50 Stone Rd E, Guelph, ON, Canada, N1G 2W1; e-mail: hassan.khan@uoguelph.ca.

ACM 2471-2566/2024/04-ART17
https://doi.org/10.1145/3648372

contextual factors are increasingly being used to reduce unnecessary **explicit authentication (EA)** while effectively defending against unauthorized access. In particular, *adaptive authentication systems* employ contextual or behavioral factors to dynamically determine whether to authenticate a user and which authentication mechanism(s) to adopt. For example, Android Smart Lock [24] keeps a device unlocked as long as the device is at a trusted location, on body, or connected to a trusted device. Several financial institutions and enterprises use contextual or behavioral factors [13, 50] to de-authenticate a suspicious user for security requirements. In academia, several adaptive authentication systems have been proposed for mobile platforms [17, 19, 37, 40, 43] to address when to and how to authenticate user based on locations, activities, behavioral biometrics, and so on.

Existing adaptive authentication solutions have the following limitations that could lead to severe problems and restrict their applications in various scenarios: **First**, most existing adaptive authentication solutions adopt a simple *single-stage* adaptation structure that can be summarized as multiple "if <*condition*> then <*adaptation*>" statements. This structure may introduce conflicts and vulnerabilities. Different contextual factors may hold simultaneously and drive opposite adaptation outcomes. For example, Android Smart Lock has conflicting interoperation between contextual factors [31]: Leaving a trusted location is expected to trigger the automatic lock, whereas a device's connection to a trusted device may make it remain unlocked. **Second,** some adaptive authentication schemes [17, 36, 37] use contextual factors to *unlock* a device, which may allow an attacker to bypass EA mechanisms. For example, Smart Lock before Android 10 [20] allowed trusted places or devices to unlock a locked device, which may be exploited by social insiders [36]. As of Android 10, Smart Lock can be used only to extend access to an unlocked device. **Third**, most adaptive authentication systems are difficult to extend or reuse [3]. However, an app may employ multiple authentication mechanisms and apply different adaptations for each. For example, a corporate app for **Bring-Your-Own-Devices (BYODs)** may adapt **implicit authentication (IA)** mechanisms to their data availability (e.g., activate gait-based IA only when a user is on foot) and tune their sensitivity based on the current location (e.g., onsite or offsite). To the best of our knowledge, there is no existing framework that supports various adaptations. We illustrate the common limitations of existing adaptive authentication solutions with two motivating examples in Section 3.

Our work is the first to explore the organization of adaptations of authentication and access control in one adaptation model and enable the deployment of adaptive authentication to handle various risks on COTS mobile devices. Upon addressing this research question, we implement a framework to enable adaptive authentication and access control for their systems and apps. Whereas MRAAC may be applicable to environments other than mobile devices, the focus of this article is on access control for mobile devices. The main challenges involve the following aspects: (1) Authentication and access control adaptations are driven by various factors and affect different components of an authentication system. Organizing them in one model is challenging. (2) An adaptation model should be extensible: it should allow adding adaptation policies for a new scenario with minimal conflicts with the existing policies. (3) An adaptive authentication system should accommodate various context factors and authentication mechanisms. Adding, removing, and modifying a component should not affect other components or the adaptation logic. (4) Integration of the adaptation framework in mobile apps and systems should introduce minimal change to the existing code.

We present a **multi-stage risk-aware adaptive authentication and access control (MRAAC)** framework (see Table 1 for comparison with existing frameworks) for mobile devices. MRAAC targets three types of stakeholders: Developers of authentication systems for mobile operating systems, security developers and administrators of **mobile device management (MDM)**

Table 1. Comparison to Existing Adaptive Authentication Solutions

		Smart Lock [24]	CASA [17]	Prog. auth. [43]	PRISM [40]	ConXsense [37]	Adaptable BA [49]	CORMORANT [19]	MRAAC
Condition	Env. factors	✓	✓	✓	✓	✓	✓	✓	✓
	User activities	✓		✓	✓				✓
	Biometrics			✓			✓	✓	✓
Outcome	Structural	✓	✓	✓	✓	✓	✓	✓	✓
	Parametric						✓	✓	✓
Access control	App-level			✓	✓	✓			✓
	In-app								✓
Structure	Single-stage	✓	✓	✓	✓	✓	✓	✓	✓
	Multi-stage								✓

"✓" denotes a supported feature. Arias-Cabarcos et al. [3] classified adaptation outcomes into structural (activation and de-activation of authentication mechanisms) and parametric (parameter tuning of authentication mechanisms).

solutions, and developers of mobile apps with high security requirements. MRAAC organizes authentication and access control adaptations in three levels (see Figure 1): The top level determines when to explicitly authenticate a user; the second level handles the adaptations driven by changes in the security context to choose an appropriate set of authenticators for the current risk type and level; the third level focuses on the adaptation of individual authenticators to ensure usability. The multi-level structure ensures that the system prioritizes security-related adaptations to prevent unauthorized access. MRAAC adopts a graph-based organization where adaptation policies are organized in multiple stages and take effect only at their affiliated stages to avoid possible conflicts with policies in other stages. The multi-stage design also enables the statefulness of MRAAC, which allows the system to progressively adjust its behaviors based on the feedback of a previous adaptation.

The contributions of our article are fourfold:

— An open-source,[1] multi-stage framework for adaptive authentication and access control for Android. Our approach supports the automatic generation of multi-stage adaptation models and enables the design of complex and stateful adaptation schemes based on contextual factors, resource sensitivity, and authentication results.
— Two libraries for developers of authentication systems and sensitive apps to enable adaptive authentication and granular in-app access control with low development overhead.
— Three use cases (an enhanced version of Android Smart Lock, guest-aware continuous authentication, and corporate app for BYOD) to demonstrate how to design multi-stage adaptation models for different use cases.
— An evaluation of MRAAC based on two implemented MRAAC-enabled apps. The performance evaluation results show that MRAAC introduces low overhead and battery consumption. A use case simulation on the public HMOG dataset [47] shows that the

[1] https://github.com/cryspuwaterloo/jiayi_thesis_code/tree/main/mraac

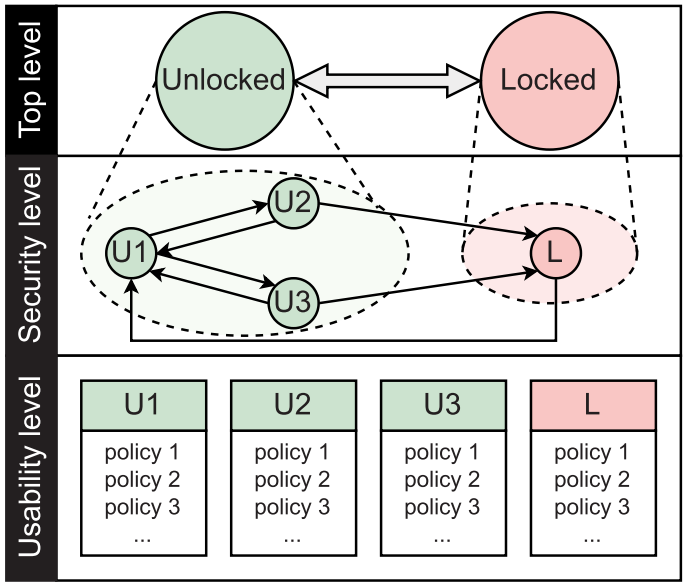

Fig. 1. Three levels of adaptive authentication and the multi-stage adaption model: (1) Top level determines whether a user should be authenticated by IA or EA and whether a user is allowed to access the device. (2) Security level manages the adaptations driven by security context changes. Stage transitions indicate the risk changes from the perspectives of authentication level, sensitivity level, and risk types. (3) Usability level manages the adaptations driven by usability context changes. Each stage hosts a set of policies to determine authentication mechanisms and their parameters.

multi-stage design can balance the false rejection rate and detection latency of continuous authentication.

2 RELATED WORK

In this section, we review existing adaptive authentication studies and compare our framework to them.

Adaptive authentication. Arias-Cabarcos et al. [3] define that an adaptive authentication system can dynamically adjust its behavior, including structural adaptations (i.e., what authentication mechanism(s) to activate) and parametric adaptations (i.e., how to tune a specific authentication mechanism), in response to the operating environment. Existing studies mainly focus on addressing how to model the context changes that trigger the adaptations [17, 37, 39, 40, 43, 48] and how to adapt authentication and access control mechanisms to satisfy the security and usability requirements of the current context [18, 37, 43, 48].

We focus on studies that provide a high-level architecture for adaptive authentication, which allows mobile app developers or device users to customize adaptation schemes. Table 1 lists the studies that satisfy the criteria and compares these studies with our adaptive authentication and access control framework, MRAAC. TreasurePhone [45] proposed "spheres" to manage access control rules based on location and user activity. It determined the visibility or availability of a certain file or app based on the current context, whereas MRAAC provides developers with a client library to enable in-app access control. Hayashi et al. [17] proposed CASA, a probabilistic framework for dynamically determining whether to explicitly authenticate a user and how to select an explicit authentication mechanism based on the user's location data. ConXsense [37] combined both

locations and nearby Bluetooth devices to calculate the familiarity of the current context and decide whether to lock the device. However, it only considered three location tags (i.e., public, work, and private) and two safety levels (i.e., safe and unsafe), which lacks extensibility. PRISM [40] proposed a context modeling approach that used HARD-BN [41] to automatically extract patterns from location, user activities, and so on. It can automatically generate policies to determine if an EA mechanism is required to unlock a device. Besides, it also allows users to manually make policies. Progressive authentication [43] employed behavioral biometrics as a contextual factor to estimate the user authenticity level. However, it did not involve the adaptation of behavioral biometrics-based IA mechanisms. Wojtowicz and Joachimiak [49] built a generic model for contextual factors that affect the performance of biometrics-based authentication mechanisms. Their main focus was on the adaptation of explicit authentication mechanisms. CORMORANT [19] is a multi-modal authentication framework that incorporates contextual factors into the fused score of several biometrics-based EA and IA mechanisms. It supports both structural and parametric adaptations of IA mechanisms and fuses multiple modalities by dynamically adjusting their weights based on contextual factors. Compared to CORMORANT, MRAAC supports a multi-stage adaptation structure to handle various risk types and includes access controls into the loop of adaptive authentication.

The context models proposed by these studies can be converted to context providers in our framework, which summarize the current context based on various sensor data (see Section 8). Compared to these studies, our focus is on proposing a new multi-stage structure for adaptive authentication. The multi-stage design of our work organizes different adaptation policies based on the risk type and uncertainty of the user's identity. Furthermore, it allows progressive adaptation that incorporates authentication results of previous adaptations.

Continuous Authentication. Our framework uses IA mechanisms to continuously verify the user's identity without their attention, which is **continuous authentication (CA)**. IA leverages users' distinct device usage [7, 15, 38] or behavioral patterns [5, 12, 26, 47, 52] to distinguish a user from others in a non-intrusive way. Another relevant technique is zero-effort authentication [34, 46], which determines whether to authenticate or de-authenticate a user based on additional devices (e.g., smartwatches, bracelets) while requiring no extra effort by the user. As the consequence of a behavioral mismatch with the owner, the current user can be blocked from accessing the device and asked for re-authentication [27]. CA can be viewed as an adaptation of determining when to authenticate a user: Detecting anomalies in the user's behaviors results in the activation of an EA mechanism. In Section 6, we demonstrate how to incorporate IA mechanisms to achieve CA and enhance the security of Android Smart Lock.

The design and development of a dynamic CA system that can adapt itself to context changes is still an open issue [14]. Some studies applied parametric adaptations to the IA/CA mechanisms based on contextual factors to achieve better accuracy or reduce power consumption. Primo et al. [39] adapted the gait authenticator to the device location and placement to improve the identification accuracy. However, this adaptation is limited to a specific authentication mechanism, while MRAAC aims to provide a generic solution to adapt various mechanisms. MultiLock [2] used facial recognition for CA and computed a confidence score from the facial recognition results. However, it only took the current app sensitivity as the factor to determine whether to lock a device. In comparison, MRAAC takes different risk types into account. As aforementioned, CORMORANT [19] dynamically adjusted its IA mechanisms to provide CA with good time coverage. The adaptation is based on a risk score calculated from location, time, and device usage. Our article demonstrates an example of applying progressive adaptation for CA, including adjusting authenticators and result aggregation at different stages. The evaluation shows that MRAAC can help balance false rejection rate and detection latency.

3 MOTIVATION

This section presents two motivating examples to show the limitations of existing adaptive authentication systems and then summarizes the requirements of a desired adaptive authentication and access control framework.

3.1 Motivating Examples

Smart Lock. To reduce unnecessary EA, Android Smart Lock [24] leverages three contextual factors to determine whether to ask for EA when a user starts a new session: (1) ***On-body Detection (BODY)***: whether the device is with a user in movements; (2) ***Trusted Places (PLACE)***: whether the current location is trusted; (3) ***Trusted Devices (DEVICE)***: whether any trusted devices is connected with the device. Smart Lock (before Android 10) uses the following five adaptation policies:

(1) If the device is not on body (`BODY == false`), then activate EA for a new session.
(2) If the current location is trusted (`PLACE == true`), then de-activate EA for a new session.
(3) If the current location is not trusted (`PLACE == false`), then activate EA for a new session.
(4) If any trusted device is connected (`DEVICE == true`), then de-activate EA for a new session.
(5) If no trusted device is connected (`DEVICE == false`), then activate EA for a new session.

Note that "activating EA" does not mean an immediate device locking, and the user can still use the device for the current session.

We observe several limitations:

(P1.1) *Policy conflicts.* If the device is connected to a trusted device in an untrusted location, then Policies 3 and 4 are satisfied simultaneously. However, the adaptation outcomes of the two policies are contradictory. Additional mitigation mechanisms need to be applied to resolve the conflict. For example, the system has to apply policy combination [42] or voting strategies [33] to prioritize one outcome over another for situations where multiple conditions hold.

(P1.2) *Misusing contextual factors for unlocking.* The contextual factors in Policies 2 and 4 result in the de-activation of EA. However, contextual factors do not provide as strong evidence as EA mechanisms that the current user is legitimate. As a result, using contextual factors to unlock a locked device makes it easier for an attacker to bypass authentication. Specifically, it introduces a higher risk of social insider attacks [36] where an attacker is in the trusted environment set by the owner and can unlock the device without having to pass EA. Therefore, Android 10 and later changed the behavior of Smart Lock to only keeping an already unlocked phone unlocked, which prevents an attacker from bypassing the EA of a locked device. However, social insiders can take advantage of the phone being kept unlocked to access the device. For example, the owner sets the workplace as trusted to keep the smartphone unlocked at work (after unlocking the phone with EA when the owner arrives at work). A co-worker can wait for the owner's temporary absence from the phone (e.g., while getting coffee) and access the unlocked device during the extended unlock interval.

(P1.3) *Stateless.* The above policies take effect regardless of the device state as long as their conditions are satisfied. It is hard to implement complex adaptations such as the "low watermark" adaptation [37]: If the three factors are all negative at the same time, then the device is considered in the insecure state. EA should be required even if either Policy 2 or Policy 4 become satisfied. The insecure state should end only after the user has passed EA.

Section 6 presents Smarter Lock to address the above problems and provides an IA-enabled option to incorporate **continuous authentication (CA)** into Smarter Lock. To further show the extensibility of our solution, we implement the guest-aware CA use case that adapts authentication and access control to guest access, which is a special scenario derived from the social insiders.

BYOD. Itus [27] is a general IA framework for BYOD apps that supports various behavioral biometrics-based IA mechanisms (e.g., Touchalytics [12], SilentSense [5]). Assume an enterprise E-mail client app is using Itus to enable IA. It uses a location contextual factor to determine if the device is within the company or not and extracts behavioral biometrics from body movements and touch events. To coordinate them, the app adopts the following adaptations:

(1) If a mismatch in behavioral biometrics (i.e., a negative IA result) is detected, then lock the device and activate EA.
(2) If the current location is not within the company (i.e., offsite), then activate IA.
(3) If the current location is within the company (i.e., onsite), then de-activate IA.

The adaptations supported by Itus mainly include two types: using IA mechanisms to determine when to activate EA and using context factors to determine when to activate a certain IA mechanism. However, they are insufficient from the following aspects:

(P2.1) *Conditions.* An adaptation can be driven by multiple reasons. As location implies the risk level of the current context (i.e., Policies 2 and 3), data availability and resource sensitivity are other possible conditions to determine whether to activate an IA mechanism. For example, if there is no significant movement, then de-activate body movement–based IA and activate keystrokes or touch events-based IA instead. Accessing sensitive resources requires stricter IA mechanisms to prevent unauthorized access.

(P2.2) *Adaptation outcomes.* The outcomes of the above policies are all about activation and de-activation. In practice, it is also feasible to tune the parameters of an authentication mechanism and change access control policies. For example, IA mechanisms for the offsite scenario can switch to a more sensitive configuration than the onsite scenario to defend against higher risk of unauthorized access. The system can also block access to specific resources when a user's identity is uncertain.

(P2.3) *False rejection.* According to Policy 1, a negative IA result immediately locks a device and requires EA. False rejection will make an authentication system less usable. It is preferrable to have an intermediate step before locking a device to mitigate false rejection while still securing sensitive resources.

In Section 6.3, we present the BYOD use case that provides a comprehensive solution to enable adaptive authentication and access control for a real E-mail client app.

3.2 Summary

The motivating examples have shown that adaptations can be driven by various factors and applied to multiple authentication mechanisms. Multiple adaptations can co-exist in the same system to handle a variety of scenarios. Unorganized adaptations can hardly fulfill the security and usability requirements. As reviewed in Section 2, no existing studies have proposed an adaptation model to organize various adaptations of authentication and access control. To address the above issues and limitations, our adaptation model is expected to:

(1) Categorize adaptations based on their conditions and organize them in a multi-level structure to avoid possible conflicts rather than mitigate them (P1.1);
(2) Model risks from the perspectives of user authenticity, resource sensitivity, and threat to describe the system states (P1.3) and cover various conditions that imply risk changes (P2.1);
(3) Regulate adaptations so attackers cannot exploit contextual factors to bypass the authentication system (P1.2);
(4) Enable progressive adaptations that adjust authentication and access control based on the feedback of a previous adaptation (P2.2) instead of directly imposing EA to mitigate false rejections (P2.3).

3.3 Threat Model

User authentication and access control defend against unauthorized access. We trust the device owner or the primary user (for corporate-owned devices). Attackers are other people who are physically near the device and can physically access it. Their goal is to access (sensitive) resources on the device.

We classify attackers into two categories based on their relationship with the owner: (1) *Strangers* are nearby attackers who have little or no knowledge of the device owner and the adaptive authentication system (e.g., what context factors result in the activation of EA). A common scenario is the device owner leaving their device unattended and a stranger picking up the device. (2) *Social insiders* are nearby attackers who are socially close to the owner. They may share a trusted environment (based on a device owner's trust perception) with the owner (e.g., a home) and take advantage of contextual factors to bypass the authentication system. For example, social insiders can launch shower-time attacks [35] where they access an unlocked device during the owner's absence. A special type of social insiders are guest users who the owner temporarily shares their device with. Guest users are allowed to access non-sensitive resources, but not sensitive resources.

Since our goal is to provide a general adaptive authentication framework, attacks on specific authentication mechanisms are not our focus. Thus, we assume attackers do not have credentials for EA mechanisms and cannot compromise IA mechanisms (e.g., in a mimicry attack [29]).

Since we provide an Android implementation of the adaptive authentication and access control framework (see Section 5), we make the following assumptions: We trust the device and its operating system, and therefore, assume that attackers cannot modify the system (e.g., tamper the sensor measurements) to deceive or bypass the authentication system. We assume that device owners acquire apps or services that adopt our proposed framework from trusted sources. However, given the framework allows communications with apps, it is possible for them to acquire sensitive resources or send false information. We discuss such attacks in Section 5.2.2 and Section 8.

4 MODELING AND ADAPTATION

In this section, we give the definition of a multi-stage adaptation model and the model construction process. Table 2 is the notation table for the multi-stage adaptation model.

4.1 Definition

Three levels of adaptive authentication. Figure 1 shows how we divide adaptive authentication into three levels based on the adaptation reasons (i.e., what leads to changes in an authentication system). The top level of adaptive authentication addresses *when to authenticate or de-authenticate a user,* which is the fundamental adaptation logic of continuous authentication. Both EA and IA assist an adaptive authentication system: EA provides mandatory identity verification when a user logs into the system, and IA continuously verifies a user's identity throughout a session and provides weak but transparent authentication for low-risk levels. Since IA rejections de-authenticate a user and activate EA [12, 27], switching between EA and IA is an adaptation process based on the user's identity.

The other two levels address how to authenticate a user based on the *security* context and the *usability* context, respectively. Changes in the security context imply changes to the risk of attacks and determine the requirements of the authentication mechanisms to handle the risk. For example, a higher risk (e.g., accessing sensitive resources) calls for a stricter set of IA mechanisms that are more sensitive to imposters. In comparison, changes in the usability context are related to the availability and performance of the authentication mechanisms, which helps determine the

Table 2. Notation Table

Name	Notation	Description
Risk types	$R = \{r_0, r_1, \ldots, r_{n_R-1}\}$	Attack risks by contextual factors; n_R is the number of risk types; e.g., $R = \{r_{\text{HOME}}, r_{\text{WORKPLACE}}\}$ includes two risk types based on user locations: HOME and WORKPLACE
Authentication levels	$A = \{a_0, a_1, \ldots, a_{n_A-1}\}$	Confidence of identifying a user as the owner; $n_A$is the number of authentication levels; e.g., $A = \{a_0, a_1, a_2\}$ represents three authentication levels: a_0 unauthenticated, a_1 low-level, a_2 high-level
Sensitivity levels	$C = \{c_0, c_1, \ldots, c_{n_C-1}\}$	Describes sensitivity of a resource being accessed; n_C is the number of sensitivity levels; e.g., $C = \{c_0, c_1, c_2\}$ represents three sensitivity levels: c_0 no resource is accessed, c_1 non-sensitive resource is accessed, c_2 sensitive resource is accessed
Authentication results	Γ	Results of authentication methods (e.g., IA acceptance/rejection)
Authentication transition	$\lambda_r : A \times \Gamma \rightarrow A$	A function that indicates how authentication results change authentication levels
Access control function	$\mu_r : A \times C \rightarrow \{\text{True}, \text{False}\}$	A function that indicates whether an authentication level is sufficient to access a resource at a given sensitivity level
Transition signals	Φ	Contextual factors and authentication results that cause risk changes
Risk transition	$\kappa : R \times \Phi \rightarrow R$	A function that indicates how risk types change upon transition signals
Input signals	Σ	All signals that cause security-level adaptations
Stages	$S = \{(r, a, c) \mid r \in R, a \in A, c \in C\}$	All stages, consisting of the combination of risk types, authentication levels, and sensitivity levels
Initial stage	s_0	The starting stage of the unlocked state
Stage transition	$\delta : S \times \Sigma \rightarrow S$	A function that determines the next stage based on an input signal and the current stage
Multi-stage model	$M = (\Sigma, S, s_0, \delta)$	A finite-state machine that describes the authentication and access control adaptation process

(de-)activation and tuning of these mechanisms. For example, low ambient brightness may result in choosing an authentication mechanism other than facial recognition. Thus, based on the scale of impact from the adaptation outcome, we define the security-driven (or risk-driven) adaptation as security-level adaptation and the usability-driven adaptation as usability-level adaptation.

Table 3. Two Examples of Authentication Transition Functions

	Default				Capped			
auth. level	IA+	IA−	EA+	EA−	IA+	IA−	EA+	EA−
a_0	−	−	a_2	a_0	−	−	a_2	a_0
a_1	a_2	a_0	−	−	a_1	a_0	−	−
a_2	a_2	a_1	−	−	a_2	a_1	−	−

Default: a negative IA result decreases the authentication level by one, while an EA acceptance raises it to the maximum. Capped: a positive IA result cannot raise the authentication level from a_1 to a_2. Since the device should be locked at a_0, IA is not activated, and therefore, IA signals (i.e., IA+ and IA-) are invalid at a_0. Similarly, EA signals (i.e., EA+ and EA-) are invalid at a_1 and a_2 when the device is unlocked.

Risk. We describe the risk from three factors: the *risk type* of the scenario, the *authentication level* of the user, and the *sensitivity level* of the accessed resource. Risk types are characterized by contextual factors, such as locations and activities, and have distinct adaptation requirements. Authentication levels indicate the confidence of identifying a user as the owner. A low authentication level calls for stricter IA mechanisms or even EA. Adopting multiple authentication levels gives flexibility when processing an IA rejection. For example, the first IA rejection may trigger further verification and restrict access to sensitive resources, instead of a direct lockout, to mitigate the impact of false rejections. Sensitivity levels and authentication levels determine whether a user can access a resource. If the authentication level fails to match the sensitivity level of the target resource, then the system rejects the access and asks for EA. Besides, an authentication system should impose stricter authentication mechanisms for access to resources with higher sensitivity levels.

4.2 Multi-stage Adaptation Model

Risk Model. We define the three risk factors as follows: the risk type set $R = \{r_0, r_1, \ldots, r_{n_R-1}\}$, the authentication level set $A = \{a_0, a_1, \ldots, a_{n_A-1}\}$, and the sensitivity level set $C = \{c_0, c_1, \ldots, c_{n_C-1}\}$, where n_R, n_A, and n_C are the sizes of the three sets. The risk type is a categorical variable that characterizes the variety of attack risks by contextual factors. An example is to define risk types based on location: If the current location is *home* or *workplace,* then apply different adaptation strategies for each of them. We reserve r_0 as the complement of all defined risk types, which represents the risk of general unauthorized attacks. Authentication levels and sensitivity levels can be expressed as two ascending sequences where $a_0 < a_1 < \ldots < a_{n_A-1}$ and $c_0 < c_1 < \ldots < c_{n_C-1}$. We reserve a_0 and c_0 for the Locked state, representing the lowest authentication level and the lowest sensitivity level, respectively.

For each risk type, we use two functions to describe the authentication and access control behaviors: (1) Authentication results lead to changes in authentication levels, and the mapping between them is a function $\lambda_r : A \times \Gamma \rightarrow A$, where Γ is the authentication result set (e.g., IA acceptance/rejection). Table 3 shows two examples of authentication transition functions. (2) Mandatory access control is a function $\mu_r : A \times C \rightarrow \{\text{True}, \text{False}\}$, which determines if the user's authentication level is insufficient to access resources at a specific sensitivity level. If the function returns False, then the user should be rejected and may need further authentication (usually EA) for a higher authentication level. The default access control function is $\mu_r(a_i, c_j) = \mathbb{1}_{i \geq j}$.

To describe transitions among risk types, we use a function $\kappa : R \times \Phi \rightarrow R$, where Φ is the set of transition signals, including contextual factors and authentication results. For example, if the location sensors detect that the device is not at a safe place, the system changes the risk type for adaptation to public locations.

Multi-stage model. The design of an adaptation model follows the three adaptation levels introduced in Section 4.1. On the top adaptation level, the adaptation model is divided into the unlocked and locked states of a binary access control model [18], where IA is activated in the unlocked state and EA is activated in the locked state. For the security-level adaptation, the system dynamically adjusts its behaviors in response to the input signals that change the current authentication level, sensitivity level, and risk type. We use a **finite-state machine (FSM)**-based multi-stage model to describe the adaptation process. Each stage (i.e., state) hosts a set of adaptation policies to apply the usability-level adaptations.

The multi-stage model M is a quadruple (Σ, S, s_0, δ): Σ is the set of all input signals for security-level adaptation, S is the stage set: each stage $s \in S$ is a combination of $(r, a, c) \in R \times A \times C$; s_0 is the initial stage for the unlocked state, which is by default (r_0, a_{n_A-1}, c_1) (i.e., general risk type, highest authentication level, low sensitivity level); $\delta : S \times \Sigma \rightarrow S$ is the transition function that determines the next stage based on an input $x \in \Sigma$ and the current stage s. We reserve the locked stage l for EA (e.g., Android Keyguard). The stage transition from l to s_0 is triggered by positive EA results (denoted by e^+): $\delta(l, e^+) = s_0$, while negative EA results (denoted by e^-) trigger a self-transition: $\delta(l, e^-) = l$.

We define that any input signal $x \in \Sigma$ may change *only one* of the three factors (r, a, c) at a time. An event that causes changes in the security context can be broken down into a series of input signals in chronological order. For example, when the system detects a guest access event, it issues an authentication signal to lower the authentication level and then produces a context signal to change the risk type. If $\delta(s, x)$ is not defined, then the system processes it as an exception (i.e., x is an invalid input at stage s) and moves to the locked stage.

The generation of a multi-stage adaptation model requires a risk model provided by developers (see target audience in Section 5.1), which specifies the following information: the risk type set R, the authentication level set A, the sensitivity set C, the authentication result set Γ, the context signal set Φ, the risk transition function κ, and the authentication and access control functions under each risk: $\{\lambda_r\}_{r \in R}$ and $\{\mu_r\}_{r \in R}$, respectively. The automatic model generation process consists of the following steps:

(1) Construct the input set $\Sigma = \Gamma \bigcup C \bigcup \Phi$ to include all valid inputs that change any of (r, a, c). Note: A sensitivity level $c \in C$, as an input, means "the user attempts to or is currently using a resource of sensitivity level = c."
(2) Obtain all valid combinations of three factors $\Omega \subset R \times A \times C$ where the authentication level is sufficient for accessing resources of the sensitivity level under the risk type: $\Omega = \bigcup_{r \in R} \{(r, a, c) | \mu_r(a, c) = \text{True}, \forall a \in A, c \in C\}$. The stage set is constructed as $S = \Omega \bigcup \{l\}$.
(3) Set the initial stage s_0 to (r_0, a_{n_A-1}, c_1) and all outgoing transitions from the Locked stage l (i.e., l to s_0 and self-transition) in δ.
(4) Generate the stage transition function δ by traversing all stages $s \in \Omega$ and inputs $x \in \Sigma$:

$$\delta(s, x) = \begin{cases} (r, \lambda_r(a, x), c) & x \in \Gamma, \\ (r, a, x) & x \in C, \mu_r(c, x) = \text{True}, \\ (\kappa(r, x), a, c) & x \in \Phi, (\kappa(r, x), a, c) \in \Omega, \\ l & \text{otherwise.} \end{cases}$$

The generated multi-stage model defines the security-level adaptation behaviors of the system. Figure 1 (security-level) shows an example multi-stage adaptation model. The stage transition process enables the system to dynamically capture the risk change and change its adaptation scheme in response to the input signals. By traversing all valid combinations of the three risk factors and all acceptable inputs, the completeness of the model is ensured to cover all possible situations. The deterministic property of stage transition (i.e., given the current stage and the input, the outcome

stage is deterministic) avoids possible ambiguity or conflicts. The stateful property provides the flexibility of adopting different reactions to the same signal. For example, a negative IA result does not always trigger de-authentication if the user is accessing a non-sensitive resource.

In Section 6, we present three use cases to show in detail how to design risk models and generate the multi-stage adaptation models accordingly.

4.3 Adaptation Policies

In a multi-stage model, each stage hosts a list of adaptation policies that determine how to change the authentication mechanisms and access controls at the usability level (see Figure 1). Adaptation policies are also configured by developers. The adaptation policies are classified into two categories based on how they are triggered: *general adaptation policies* and *conditional adaptation policies*. (1) General adaptation policies take effect once a stage transition happens. They determine the default authenticators and access control rules of a stage. For authentication adaptation, the system activates the authentication mechanisms listed in the general adaptation policies of the current stage. It also deactivates the ones that are activated in the previous stage but not listed in the current stage. For access control, the multi-stage adaptation model determines resource availability based on sensitivity levels and authentication levels. The access control policies within each stage can further adjust the availability of individual resources. (2) Conditional adaptation policies are a set of "if-then" statements—once the conditions are satisfied, the corresponding adaptation takes effect. They mainly handle the usability-level authentication adaptation process to determine when to activate and deactivate authentication mechanisms and how to tune the parameters in response to the usability context (e.g., if *no significant movement is detected*, then *deactivate gait authenticator*.)

The multi-stage adaptation model also supports the design of the adaptation policies. By comparing the risk type, authentication level, and sensitivity level, one can determine if a stage is "riskier" than its neighboring stages (i.e., stages connected to it by a transition edge). A stage at higher risk should have no less strict/less secure authentication and no more available resources than a stage at lower risk. Following the security metrics of biometric-based authentication mechanisms [23], developers can compare the authentication mechanisms between two stages and determine if the proposed adaptation policies are reasonable.

4.4 More Complex Access-control Models

Our access-control model grants a user access to a resource if the user's authentication level is sufficiently high (see Table 2). There are more complex access-control models, such as RBAC and ABAC, where users are assigned to roles/attributes and where access control ensures that a user has a particular active role/attribute. CA-ARBAC [1], DR BACA [44], and CAPEF [25] propose context-aware RBAC and ABAC models for Android. However, access control in these papers focuses on deciding whether an app, not a user, should be granted a permission to an Android resource, such as the network. In turn, roles and attributes are assigned to apps, not to users. In MRAAC, access control is about deciding whether a user should be granted access to a device (i.e., whether the user should be allowed to unlock the device), to an app installed on the device, or to individual components of an app. Therefore, these existing works and MRAAC are largely orthogonal.

Whereas it is possible to add support for RBAC and ABAC to MRAAC (i.e., the access control function in Table 2 would depend on a user's roles or attributes, not on the user's authentication level), we believe that the additional complexity does not warrant this effort given that a mobile device is often used by only a small number of people. Moreover, attributes, such as contextual factors, can be modeled as risk types in MRAAC (see Section 6.3 for a use case based on location). MRAAC has the advantage that transitions between risk types are state-based and explicitly

modeled. This avoids problems due to misuse of contextual factors, where an attacker simply by changing a device's context is granted additional access rights (see the Smart Lock example in Section 3.1).

5 SYSTEM DESIGN

5.1 Target Audience

We design MRAAC to provide a general adaptation framework for authentication and access control on mobile devices and apps. Our audiences are: (1) security developers of OS vendors and mobile authentication systems, (2) security developers and administrators of **mobile device management (MDM)** and **enterprise mobility management (EMM)** solutions, and (3) developers of sensitive apps.

Authentication system developers can use MRAAC to manage the adaptation process to enhance their systems and access control to sensitive resources. Similarly, MDM/EMM solutions require an endpoint management agent or are integrated with the operating system (e.g., Android Enterprise) to collect contextual information and enforce security policies. These security developers and administrators are also responsible for making multi-stage adaptation models and designing security policies for the system.

As a mobile authentication system provides authentication APIs (e.g., Android's BiometricManager) for apps, MRAAC also enables risk awareness for in-app adaptation. We propose a "client-server" structure for MRAAC: the server is a centralized service that performs risk evaluation and provides system-wide adaptive authentication and app-level access control, and the clients are apps that receive signals from the server to enable risk awareness and adjust the availability of in-app resources accordingly. Since the server handles most operations, clients can obtain the digested risk information without accessing sensitive permissions or performing expensive computations. Developers of sensitive apps can focus on specific high-risk stages (e.g., non-owner access) and in-app resources that are not controlled by the system (e.g., terminating an app session or invalidating app tokens). In this case, they do not need full knowledge of the multi-stage adaptation models running on the target device.

Nevertheless, individual, sensitive apps may require custom adaptive authentication and access control even if a device does not provide a system-wide MRAAC service. For example, corporate apps for BYODs require dedicated adaptive authentication systems to manage user access based on location and network. However, we cannot expect the presence of a system-wide MRAAC service on user-managed devices. For this deployment scenario, MRAAC needs to be integrated entirely (i.e., both its client and server part) into these sensitive apps (called *host apps*) to enable risk awareness and adaptation management.

Users typically cannot modify adaptation models or security policies on their devices. However, they can have the flexibility of model configuration, such as choosing a preferred explicit authentication method to unlock their devices.

5.2 Architecture

Figure 2 shows the architecture of MRAAC in the client-server design, which consists of a system-wide service ("MRAAC Service") and a client library ("MRAAC Client"). Note that the server is deployed locally on mobile devices as an Android service. For the individual apps deployment scenario, we propose the MRAAC Integration library to include the components of both MRAAC Service and MRAAC Client.

5.2.1 MRAAC Service. MRAAC Service provides context detection and adapts authentication mechanisms and access control. It consists of four modules, and each module contains a service

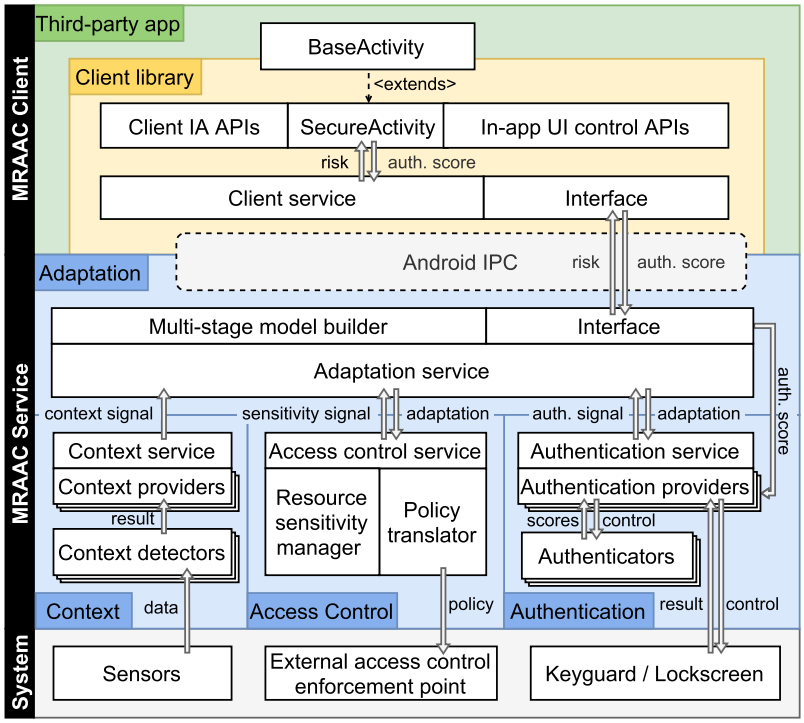

Fig. 2. System design of the MRAAC framework.

handling the communication with other modules: **Adaptation module** is the core of MRAAC. It provides the multi-stage model builder to help developers generate adaptation models and design adaptation schemes. As introduced in Section 4.2, a multi-stage adaptation model defines all stages and transitions. With the model, the adaptation service can manage the adaptation process: It listens to authentication results, sensitivity changes, and contextual signals from the other modules. According to the stage transition table, the adaptation service performs stage transitions and enforces adaptation policies based on the current stage and the received signal. It sends the adaptation outcome as a control signal to the other modules for changing their behaviors. In addition, the adaptation module provides an interface to handle communications with MRAAC client apps.

Context module hosts a list of context detectors to collect and process data from various sensors and generate contextual factors used in adaptation conditions. Each context detector works independently as a service, and a context provider is bound to it to convert its results into context signals. The context service manages a list of context providers and forwards context signals to the adaptation service. During the initialization, the context service activates all context detectors via the control interface provided by the corresponding providers.

Authentication module manages and adapts all authentication mechanisms. Similar to the context module, each authenticator runs as a service and binds to an authentication provider. The authentication service aggregates results from authenticators and sends authentication signals to the adaptation module. Besides, it receives adaptation signals from the adaptation service to (de-)activate or adjust the target authenticators or tune its aggregation method. Note that EA mechanisms provided by the operating system (e.g., Android Keyguard) also require an authentication

provider as an interface for activation and receiving authentication results. In addition, a client app may have its own authenticators (see Section 5.2.2). We can also instantiate providers for client authenticators to receive their scores and involve them in the adaptation loop.

Access control module determines the current sensitivity level and manages access control policies. The access control service tracks the current resource and notifies the adaptation service of sensitivity changes. The resource sensitivity manager maps resources to sensitivity levels for each stage. The enforcement of access control policies is performed by the adaptation service and the external access control enforcement point: (1) the adaptation service determines whether to move to the locked stage after receiving the sensitivity change; (2) the access control service receives the adaptation scheme of the current stage from the adaptation service, and then the policy translator translates each access control rule into the policy format accepted by the external access control enforcement point (e.g., FlaskDroid [8], ASF [4]).

5.2.2 MRAAC Client. Given that an app may contain resources at different sensitivity levels, it is more flexible to control access to in-app resources than to fully block access to the app [18]. Instead of offloading context detection to apps, a better approach is to enable apps to receive contextual cues from the adaptive authentication system through a library. Client apps are the apps that depend on MRAAC Service to obtain the stage information for the adaptation of in-app authentication and access control. MRAAC provides a client library for client apps, which consists of the following components:

Client service manages communications with MRAAC Service and parses the stage information. The service is activated at the startup of the app and automatically sets up the connections to MRAAC Service and pulls the current stage information. While the client app is running, the service keeps listening to broadcasts regarding stage changes from MRAAC Service. Besides, if the app adopts the IA mechanisms provided by the library, then the client service can forward authentication results to MRAAC Service via Android IPC. Sending authentication results is a security-sensitive operation, since malicious apps may send false results to deceive the system. MRAAC only allows forwarding authentication results when the identity of the client app is verified or the client app is signed with the same key as the service.

SecureActivity is a generic Android application component that client apps are supposed to extend to easily receive the stage change signal. The client library provides optional IA APIs based on the generic IA framework Itus [27] to enable IA within the app scope. It also provides UI control APIs to support in-app access control. App developers can delegate the control of specified UI elements to the `SecureActivity` so it can automatically adjust the availability and visibility of UI elements based on the current stage.

5.2.3 MRAAC Integration. We provide the MRAAC Integration library to enable standalone adaptive authentication and access control for individual apps. It enables third-party apps to integrate the full MRAAC Service and `SecureActivity` to provide a complete adaptation workflow within the host app. The locked stage means that the user needs to pass EA to continue using the app. The authentication process of the associated online service of the host app can be involved in the adaptation loop. For example, assuming that the host app is associated with an OAuth2 service [16], the locked stage for the host app can trigger the revocation of the authentication token. As a result, the host app logs the user out to secure the user's personal information. For the access control module, the host app needs to provide the currently accessed resource (e.g., view, file) to the service. Besides, in-app access control replaces the external access control enforcement point of MRAAC Service.

5.3 Development Workflow

This section lists the development workflow of using MRAAC to enable adaptive authentication and access control for MDM/EMM solutions and individual apps.

The first step is to generate the multi-stage adaptation model, which determines the high-level adaptation behaviors of the system. Security developers and administrators need to specify authentication levels, sensitivity levels, and risk types based on the security requirements of their projects. This information can be extracted from existing security policies (note: conversion from security policies and adaptation model elements is out of the scope of this article), while MRAAC also provides pre-defined settings. Then, they define the authentication and access control functions for each risk type in table form (see Table 3). `MultiStageModelBuilder` of MRAAC provides several common functions to reduce development efforts. They also need to provide all possible context signals (related to the risk type transitions) and the risk transition functions, which can be also migrated from existing security policies. Once the above information is prepared, `MultiStageModelBuilder` constructs the multi-stage model based on a configuration file (see Figure 2). Figures 4 and 6 show two example configuration files.

The second step is to set up the four main modules of MRAAC Service. MRAAC ships with a number of common authenticators (e.g., touch, gait) and context detectors (e.g., locations, activities) so developers can choose from them and rapidly build their adaptive authentication system. Given that developers may have their own implementation of authenticators and context detectors, MRAAC provides two abstract classes `BaseAuthenticator` and `BaseContextDetector` to define the essential methods required for communication and controlling their customized components. Then, developers need to register all selected authenticators and context detectors in the authentication service and context service via `addAuthenticator()` and `addContextProvider()`, respectively. For the access control service, developers need to override the `acquireCurrentResource()` method to provide the current resource and direct the output of the policy translator to their access control enforcement point. For the adaption module, the developer only needs to load the model generated in the first step without touching other parts.

The third step is to make adaptation schemes and enable MRAAC Service. According to Section 4.3, each `Scheme` object involves two kinds of adaptation policies: (1) `defaultAdaptation` takes effect at each stage transition, and (2) `conditionalAdaptation` takes effect when the condition is satisfied. Developers need to make adaptation schemes for each stage, and it is possible to set the same scheme for several stages. After scheme making, developers register all the services in the Android manifest file and start them at app/system startup.

Development workflow for client apps. Developers only need to make their base activities extend the `SecureActivity` and override `onStageChanged()` to implement their adaptation solution. The activity provides the `getCurrentStage()` method for developers to actively acquire the current stage information. App developers can change the visibility of view objects and block access to sensitive methods (e.g., methods related to file operations) to protect user data privacy and security. App developers can delegate the UI control to the client library by registering target view objects for automatic control. Since the MRAAC Integration library includes the components of the client library, host app developers can enable in-app control in the same way.

6 USE CASES

This section presents three use cases for MRAAC: Smarter Lock, guest-aware CA, and corporate app for BYOD. We adopt the following naming convention for stages: Except for the locked stage denoted by L, a stage name consists of three characters representing risk type, authentication level, and sensitivity level, respectively. For example, Stage $A21$ means that a user whose authentication

level is two is accessing the resource at sensitivity level one under the risk type A. We use the following common settings for the risk models of the three use cases: the authentication transition functions follow the default scheme in Table 3, and we adopt the default access control function (i.e., geq): To access a resource of c_i, the authentication level a_j should satisfy $j \geq i$.

6.1 Smarter Lock

We first present Smarter Lock to show how to use MRAAC to improve Android Smart Lock while addressing the issues and limitations mentioned in Section 3.1. The basic idea of Smart Lock (Android 10 and later) is that once a device is unlocked with EA, the device is kept unlocked as long as it is in a trusted environment. Namely, if any of BODY, PLACE, and DEVICE (see Section 3.1) is positive, then the device remains unlocked. However, if all three factors are negative at any moment, then the device will activate EA for the next session. Negative factors will not lead to an immediate device lock if the device is currently unlocked.

We present this use case in two steps. In the first step, the adaptation model adopts only the contextual factors used by Smart Lock (i.e., BODY, PLACE, and DEVICE) to address when to end the unlocked state of a device and activate EA.

In the second step, we show the extensibility of MRAAC by handling social insiders that are not covered by Smart Lock (including Android 10 and later). Social insiders may take advantage of the unlocked state extended by contextual factors to access the device as introduced in Section 3.1. As a countermeasure, we enable behavioral biometrics-based IA to proactively defend against social insiders. In particular, IA is used to immediately lock out an attacker upon behavioral mismatches.

Risk model. We adopt a minimal risk model for Smarter Lock with two authentication levels: a_0 (unauthenticated) and a_1 (authenticated); two sensitivity levels: c_0 (no resource being accessed) and c_1 (resource being accessed); and two risk types: Trusted (device in a trusted environment) and Untrusted (device in an untrusted environment). We use a high-level context provider **`TrustedContextProvider` (TC)** to aggregate the results of the three contextual factors. It also adopts a **timer (TIMEOUT)** to track if the duration of being in a trusted environment exceeds a predefined value. The timeout interval is configurable by the device owner. The resulting aggregation can be expressed as the following logic expression:

$$TC = (\text{BODY} \vee \text{PLACE} \vee \text{DEVICE}) \wedge \neg\text{TIMEOUT}.$$

`TrustedContextProvider` outputs two signals: `TRUSTED` and `UNTRUSTED` for transition between the two risk types. When `UNTRUSTED` is issued, the risk type changes from Trusted to Untrusted. However, `TRUSTED` cannot change the current risk type from Untrusted to Trusted, since Smarter Lock adopts the low watermark strategy. Another signal is `SCREEN_ACTION`, which indicates if any screen action is performed (i.e., screen on or screen off). This signal implies a user starts or stops using the device, and the system needs to determine whether to activate EA. For Android, it relies on a system event context provider that listens to `ACTION_SCREEN_ON` and `ACTION_SCREEN_OFF`.

Multi-stage model. Based on the risk model, we obtain a three-stage adaptation model as shown in Figure 3.

According to the model (without IA), the device remains unlocked if it stays in a trusted environment, i.e., $T11$. When `TrustedContextProvider` issues an `UNTRUSTED` signal (i.e., untrusted environment or the time limit exceeded), the current stage changes to $U11$ ($T11 \rightarrow U11$), where the user needs to pass EA for the next session. After that, the device cannot move back to $T11$ even if it receives the `TRUSTED` signal (i.e., low watermark). To add or remove a contextual factor for trusted environments, a developer needs to only change `TrustedContextProvider` without changing the topology of the multi-stage model, which helps reduce possible conflicts.

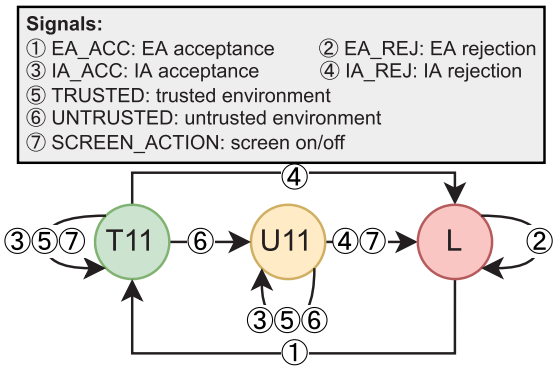

Fig. 3. Multi-stage adaptation model for Smarter Lock. Stages: T: Trusted, U: Untrusted, L: Locked. Note: Signals ③④ are available only when IA is adopted.

Adaptation. Since only EA is available in the first step, the adaptation is about whether to activate EA. To avoid misusing contextual factors for unlocking, MRAAC makes EA acceptance the only signal that can let the system leave the locked stage (i.e., $L \rightarrow T11$). It also shows that the statefulness of MRAAC makes it simple to track the state of the system and make adaptations based on both the current stage and the input signal.

Incorporating IA. Smarter Lock can employ behavioral biometrics-based IA mechanisms in Stages $T11$ and $U11$ to continuously defend against unauthorized access from social insiders. We enable the IA mechanisms and add adaptation policies to activate them at $T11$ and $U11$. If there is a negative IA result, then the system will immediately lock the device and activate EA (i.e., $T11 \rightarrow L$ and $U11 \rightarrow L$). Incorporating IA can help defend against knowledgeable social insiders: Even if they know the device remains unlocked in a trusted environment and find a chance to access the unlocked device, IA mechanisms can still block them. However, false IA rejections will immediately block a legitimate user, which may affect usability. The next use case illustrates how we use MRAAC to mitigate false rejections.

6.2 Guest-aware CA

Enabling IA to continuously authenticate a user can proactively defend against unauthorized access. However, the first use case shows the necessity of adapting IA mechanisms to mitigate false IA results. Also, it is possible for a guest user to temporarily access the device, who should not be blocked by IA. Thus, in this use case, we use MRAAC to implement a guest-aware CA system covering the following four aspects: (1) using the current resource sensitivity to dynamically adjust the parameters to balance false rejection and false acceptance rates; (2) incorporating access restrictions as a reaction to biometric mismatch; (3) activating authenticators only when the corresponding biometric traits are available; and (4) not blocking a guest user upon biometric mismatches while restricting a guest's access to sensitive resources.

Risk model. Figure 4(a) shows the configuration file of the risk model. Compared to the Smarter Lock use case, we add authentication level a_2 and sensitivity level c_2 and use the following common settings: resources are classified into non-sensitive (c_1) and sensitive (c_2), and the two authentication levels a_1 and a_2 represent weakly and strongly authenticated users, respectively. Guest-aware CA is supposed to handle the *general* risk type (i.e., general unauthorized access attacks) and the *guest* risk type (i.e., authorized access from a guest user). A `GuestContextProvider` detects specific actions that indicate a user change (e.g., Android's Screen Pinning [21]) or a device handover gesture [9] and issues the `GUEST` signal to enter the *guest* risk type. We note that guest access also

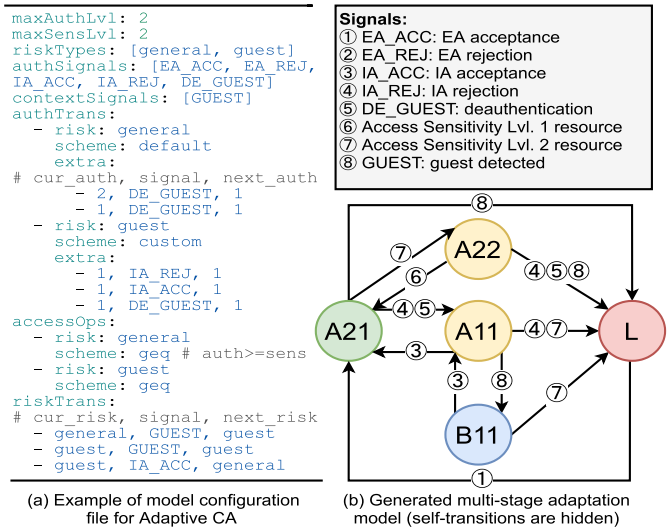

Fig. 4. Model generation for Guest-aware CA. *A*: general, *B*: guest. Stages with different colors adopt different adaptation schemes.

implies a change in authentication level, since the user changes to a non-owner. Thus, the guest context provider needs to issue an authentication signal `DE_GUEST` ahead of the context-related `GUEST` signal, which de-authenticates the user from a_2 to a_1 for guest access. In addition to the default scheme in Table 3, we add two `DE_GUEST` related transitions to the authentication transition function `authTran` of the general risk type. As for the `authTran` function of the guest risk type, the authentication level is fixed to a_1 in response to all the signals except for `IA_ACC`. Because an IA acceptance implies that the user changes back to the owner, we use it as the context signal for exiting from the guest risk type to the general risk type.

Multi-stage model. In Figure 4(b), the `MultiStageModelBuilder` builds a five-stage model based on the risk model: From the model, we can observe several adaptation flows: (1) Sensitive resource access: Only a strongly authenticated user is allowed to access sensitive resources ($A21 \rightarrow A22$), and the system blocks a weakly authenticated user or a guest ($A11 \rightarrow L$, $B11 \rightarrow L$). In addition, IA rejection or switching to a guest will trigger EA during the access ($A22 \rightarrow L$). (2) IA rejection: to mitigate false IA rejection, $A11$ acts as a buffer for further verifying the user's identity ($A21 \rightarrow A11$) and only restricting access to sensitive resources instead of a direct lockout. If IA in $A11$ accepts the user, then the system automatically addresses the previous false reject ($A11 \rightarrow A21$). (3) Guest access: guest access context produces two signals `DE_GUEST` and `GUEST` and results in two-hop adaptation flows: $A21 \rightarrow A11 \rightarrow B11$ and $A11 \rightarrow A11 \rightarrow B11$. The system downgrades the authentication level in the first hop to prevent a guest from accessing sensitive resources (i.e., $A22 \rightarrow L$). In the second hop, the system adapts to the guest risk type. When the guest finishes using the device, the current user changes back to the owner, which is captured by a positive IA result ($B11 \rightarrow A11$).

Adaptation. As MRAAC generates the multi-stage model, it provides a list of valid stages and allows developers to further specify adaptation policies for the usability-level adaptation in each stage. Assume that gait and touch authenticators are available. Given the low power consumption of touch-based IA [27], we activate it for all stages except for *L*. The default adaptation policy is expressed as a tuple `(TOUCH_IA, start, default)`, which means activating the touch authenticator with the default parameters. For the guest stage, we still need the touch authenticator to

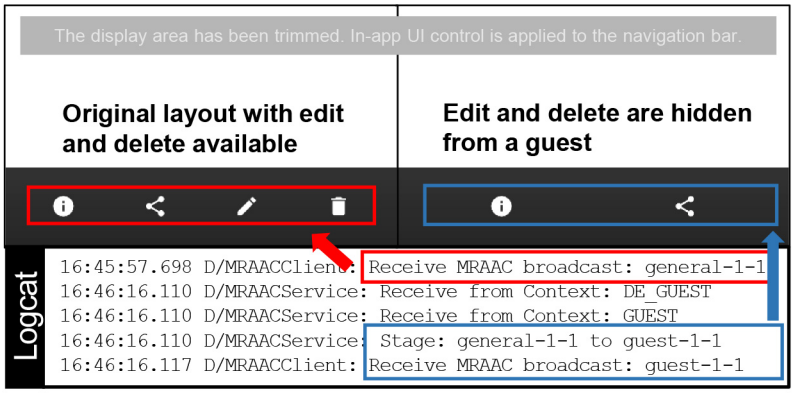

Fig. 5. Demo: MRAAC Client Camera Roll.

track the user's identity to determine when to exit from the guest stage. For higher-risk stages (i.e., $A11$ and $A22$), we additionally adopt the gait authenticator, which is activated when the user is on foot. We set up an `OnFootContextProvider` (based on the Google Activity Recognition API) to provide the context signal `ON_FOOT`. The conditional adaptation policy is `ON_FOOT` $\rightarrow$ `(GAIT_IA, start, default)`. We adopt a sliding window strategy [12, 32, 51] for result aggregation: If m out of n instances are accepted as the owner's, then the authenticator will accept the current user as the owner. An example policy is `(AUTH_AGG, tune, "--reset --m=3 --n=5")`, which means resetting the sliding window and set $m = 3, n = 5$. As for access control adaptation, the device user can customize an allowlist or a blocklist for the guest stage to determine what can or cannot be accessed by a guest user.

Demonstrative example. MRAAC Service is supposed to be a system-wide service that enables guest-aware CA for the device authentication systems. We add all authenticators and context detectors and load the risk model and the adaptation schemes to initialize MRAAC Service. To show how it works with third-party apps with MRAAC Client, we modified an open-source photo gallery app, Camera Roll [30] (see Figure 5). Our goal is to hide the file operation buttons, *delete* and *edit* at the guest risk type. Most changes were made to the app's activity classes. `ItemActivity` provides a single photo view with file operation buttons. Client app developers only need to import the MRAAC Client library and make activities inherit from `SecureActivity`. Internally, it automatically starts Client Service to connect to MRAAC Service, sends IA results, and receives the stage updates. In `ItemActivity`, we overrode `onStageChanged()` and set the visibility of target buttons to `View.GONE` at the guest risk type. In the `onCreate` method of the main activity, we added `acquireRiskType()` to pull the current risk type for initialization.

6.3 Corporate App for BYOD

Companies adopt BYOD policies to authorize employees to use their own devices for work purposes. Employees need to use a corporate app to access corporate resources. It is essential to deploy an app-wide authentication solution to secure sensitive data. In addition to EA for login, corporate apps and services also need continuous authentication to determine when to de-authenticate a user upon suspect unauthorized access during a session. Besides, as discussed in Section 3.1, existing IA frameworks are insufficient when it comes to a complicated adaptation model: The authentication solution should adopt different strategies according to whether a user is accessing the app onsite or offsite. If the user is accessing the app within the company, which is considered secure, then

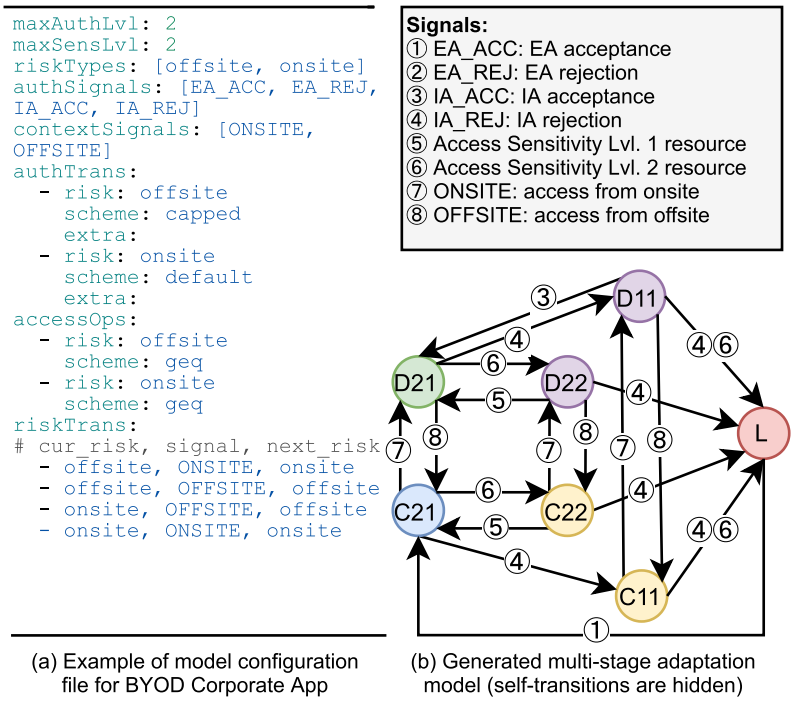

Fig. 6. Model generation for BYOD corporate app. C: offsite, D: onsite. Stages with different colors adopt different adaptation schemes.

it is unnecessary to adopt more stringent IA mechanisms; otherwise, if the user is offsite, then stricter IA is required, and more restrictions are imposed on access to corporate resources. For this use case, we use the MRAAC Integration library to enable self-contained MRAAC services and demonstrate it with FairEmail, a popular open-source e-mail client [6].

Risk model. Figure 6(a) shows the risk model for the BYOD use case. We adopt the same settings for authentication levels and sensitivity levels as the guest-aware CA use case and introduce two risk types: offsite (C) and onsite (D). We adopt the capped authentication transition function (see Table 3) for the offsite risk type—a positive IA result cannot raise a low authentication level to the maximum. It ensures that a weakly authenticated user must be explicitly authenticated to access sensitive resources. At the same time, the user can still access non-sensitive resources in the app. We adopt a location-based context provider `OnsiteContextProvider` to determine if the device is in the company or not. It generates two context signals, `ONSITE` and `OFFSITE`, to switch between the two risk types.

Multi-stage model. Figure 6(a) shows the multi-stage model generated from the above risk model. We can see that most stage transitions are identical between the two risk types except that it is impossible to move from $C11$ back to $C21$ because of the capped authentication function. The other parts are similar to the general risk type of the guest-aware CA use case.

Adaptation. Authentication adaptation involves two aspects: (1) Activation. Given that the onsite context is secure, we only need to activate the touch-based authenticator. As the offsite context implies a higher risk of unauthorized access and a user's mobility, we additionally activate the gait-based authenticator using the same conditional adaptation policy in the Guest-aware CA use case. (2) Aggregation. Since the sliding window strategy helps balance the false rejection rate and false

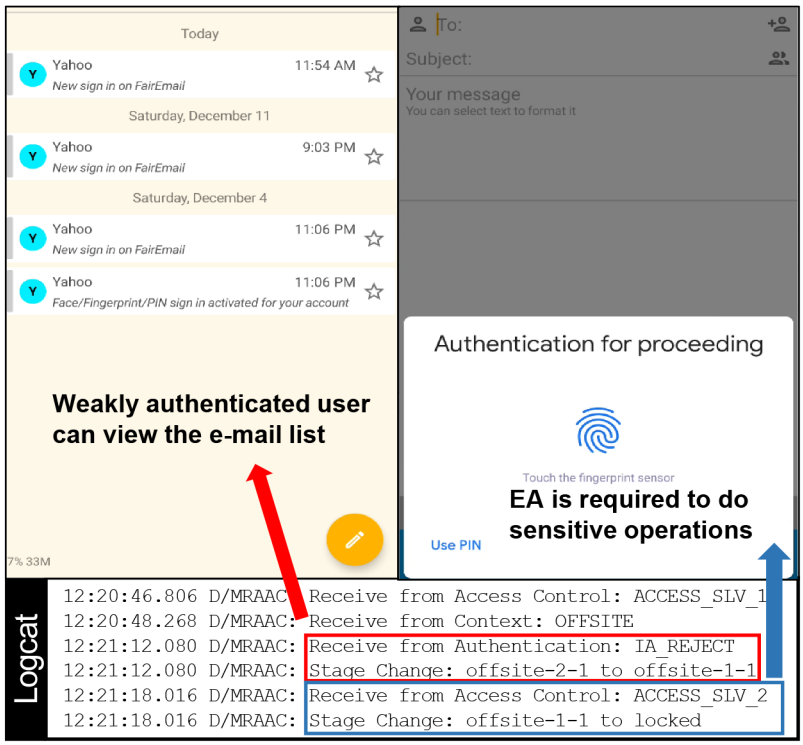

Fig. 7. Demo: MRAAC Integrated FairEmail.

acceptance rate, we adopt a larger window for the onsite risk type to reduce false rejections and a smaller window for the offsite risk type for a low false acceptance rate. Moreover, we can further adapt the aggregation strategy to each stage within a risk type. The access control adaptation scheme for the offsite risk type is mainly about adjusting the sensitivity level of specified resources based on the current stage. For example, it is considered non-sensitive to use a corporate e-mail address to send e-mails in the company. However, it becomes sensitive for the offsite scenario because of possible imposter attacks. Thus, we can raise its sensitivity level to c_2 to require a high authentication level.

MRAAC Integrated FairEmail [6]. We implemented the BYOD use case on FairEmail using the MRAAC Integration library. As introduced in Section 5.3, we created the four main services to inherit from the base services. In the adaptation service, we loaded the configuration file of the risk model and set up adaptation schemes for all stages. All the context providers and the authenticators (together with the aggregation method) should be registered in the context and authentication services, respectively. The access control service requires a resource sensitivity map for each stage so the service can update the sensitivity accordingly. Similar to the MRAAC Client library, we extended the base activity from `SecureActivity` to enable MRAAC. The difference is that we added `reportResourceName()` to the callbacks related to resource changes so the access control service can receive the current resource. Developers can also report the current file, document, or data and assign a sensitivity level. For example, the e-mail app can report the details (e.g., tag, sender) of the current message. Then, e-mails from specific senders can be assigned with a higher sensitivity level. With the above process, we integrate MRAAC into the FairEmail app. Figure 7 shows how we enabled in-app access control of FairEmail. As described in the access control adaptation

scheme, we adjusted the sensitivity level of the compose activity as high for the offsite scenario. As shown in the figure, a weakly authenticated user can view the inbox list. However, if the user wants to compose a new e-mail, they need to provide their fingerprint to proceed.

7 EVALUATION

7.1 Evaluation Setup

Devices. To show the compatibility of MRAAC, we used three different Android smartphones for performance evaluation to cover a variety of hardware and software: 1) Google Pixel (2016, CPU: 2*2.15GHz + 2 *1.6GHz, RAM: 4GB, Android 9.0), 2) Samsung S8 (2017, CPU: 4*2.3GHz + 4*1.7GHz, RAM: 4GB, Android 7.0), 3) Google Pixel 3 (2018, CPU: 4*2.5GHz + 4*1.6GHz, RAM: 4GB, Android 12). We also tested MRAAC successfully on the latest Android version to date (Android 14) using the Google Pixel 7 Pro emulator.

Performance metrics. For general performance evaluation, we measured the CPU time of critical operations and the memory overhead in terms of heap size. We also measured the inter-service communication latency to evaluate how fast the adaptation service can react to a signal. In addition, we evaluated the battery consumption of MRAAC.

Configurations. Our evaluation covered the use cases in Section 6. We evaluated the performance of the MRAAC Service integrated into the FairEmail app and evaluated the MRAAC Client using the demo Camera Roll app. *Authenticators:* We used touch- and gait-based IA mechanisms, since they rely on common sensors (i.e., touchscreen, accelerometer, gyroscope) available on most smartphones and are related to common activities (i.e., swiping, walking). The touch-based IA adopts the Touchalytics [12] algorithm and conducts classification over each touch event. The gait-based IA adopts a deep neural network–based model [52] and conducts authentication every five seconds (the sampling rate of the motion sensors is 50 Hz). For result aggregation, we aligned the results from two authenticators in chronological order and applied a sliding window for decision-making. *Context detectors:* We adopted a threshold-based `OnFootService` to tell if the user is walking. It issues a context signal every 15 seconds. The `ON_FOOT_ENTER` signal activates the gait-based IA while `ON_FOOT_EXIT` triggers the de-activation. We deployed a location context provider for the BYOD use case to detect if the user is onsite. The location update frequency was once per 15 seconds. For the Guest-aware CA use case, we manually triggered the `GUEST` signal to measure the client-server communication latency.

7.2 Development Overhead

To show that MRAAC provides a rapid development of adaptive authentication, we measured the development overhead of MRAAC Service and Client in terms of **lines of code (LOC)** for the modified FairEmail and Camera Roll apps. All code changes are in Java. For FairEmail, we count the LOC of four main services of MRAAC Service: (1) Adaptation service: 25, (2) Authentication service: 51, (3) Access control service: 33, (4) Context service: 32. We added 22 lines in the base activity of FairEmail to control the four services and enable in-app access control. To bind an existing authenticator or context detector to MRAAC, developers need to implement an authentication provider or context provider. The LOC of the provider for the gait authenticator is 24, and the LOC of the provider for the onsite context detector is 30. For Camera Roll, which only enables the MRAAC Client, there are only 15 lines added to receive the risk changes and adapt the UI components.

7.3 Performance Evaluation Results

CPU overhead. The critical operations of MRAAC are related to the adaptation process. Thus, we instrumented the adaptation service to measure the CPU time of the following aspects and

Table 4. Mean Latency (Standard Dev. in Parentheses) of Inter-service and Inter-process Communication

	inter-service			inter-process		
devices	authentication (ms)	context (ms)	adaptation (ms)	pull (ms)	broadcast (ms)	IA update (ms)
Pixel	1.15 (0.51)	1.23 (0.53)	10.20 (8.74)	2.17 (1.02)	9.74 (5.09)	2.18 (0.88)
S8	0.71 (0.62)	0.40 (0.12)	5.04 (1.45)	1.61 (0.64)	7.32 (3.65)	1.69 (0.75)
Pixel 3	0.50 (0.15)	0.56 (0.14)	2.70 (0.66)	0.58 (0.09)	4.30 (0.73)	1.53 (0.25)

repeat 10 times to calculate the average: (1) Model construction overhead. The adaptation service first needs to build the multi-stage model from the configuration file for initialization, which is a one-time operation. The results show that the average CPU time was about 35.0 ms (std: 0.3 ms) for Google Pixel 3, while Google Pixel needed 111.9 ms (std: 6.4 ms). (2) Low-level adaptation overhead. We measured the CPU time of the adaptation service processing a low-level adaptation, which was using the `ON_FOOT_ENTER` signal to activate the gait authenticator. Most cases were below 1 ms for Google Pixel 3 and Samsung S8. Even for Google Pixel, the adaptation overhead was only 1.6 ms (std: 0.1 ms), which is negligible. (3) Stage transition overhead. We tested with the stage transition $C21 \rightarrow D21$ triggered by the `ONSITE` context signal in the BYOD use case, which involves enforcing three default adaptation policies (i.e., low-level adaptation): de-activating the gait authenticator, activating the touch authenticator, and adjusting the sliding window for aggregation. The results show that the CPU time of a stage transition was about 3.3 ms (std: 0.1 ms) for Google Pixel 3 and about 8.6 ms (std: 0.8 ms) for Google Pixel. Given that all stage transitions were pre-computed during model construction, it took low CPU overhead to perform stage transitions. Based on the low-level adaptation overhead, we can see that the enforcement of default adaptation policies took the most overhead in the stage transition.

Memory overhead. We used the Android Profiler to measure the heap size of these services. Note that we excluded the extra memory overhead from each authenticator, given that developers can choose different sets of authenticators and may produce very different results. Specifically, machine learning models and buffered sensor data may take a significant amount of memory space. The results show that MRAAC Service introduced 22 kB memory overhead and MRAAC Client introduced only 2 kB memory overhead. For comparison, the motion data buffer (250 samples with six double variables for each sample) used by the gait authenticator was 12 kB.

Latency analysis. Compared to single-stage schemes, MRAAC requires additional operations to handle multi-stage adaptation, which results in an increase in latency. Besides, authenticators and context detectors, as services, need to communicate with the authentication service or the context service to reach the adaptation service. Thus, we conducted latency analysis over each step of MRAAC to measure their impact on the time overhead.

MRAAC adopts an event bus for all signal exchanges among internal services. We measured the following inter-service latency types and repeated each experiment 100 times: (1) authentication latency: the elapsed time from when an authenticator sends an authentication score to when the adaptation service receives the signal. (2) context latency: the elapsed time from when a context detector sends a context signal to when the adaptation service receives the signal. (3) adaptation latency: the total elapsed time from when a context detector generates a context signal to when the target authenticator receives the adaptation signal (i.e., context detector → context service → adaptation service → authentication service → authenticator). Table 4 shows that both the authentication latency and the context latency were only around 1 ms for all three phone models. For newer phones like Google Pixel 3, they were always below 1 ms. The adaptation latency was longer, since it involved more hops and adaptation policy processing. Nevertheless, even for Google Pixel, the average adaptation latency was around 10 ms, which means the authenticator can adapt to the context change within a negligible time interval.

In the client-server structure, an MRAAC Client app communicates with MRAAC Service via the Android IPC mechanism to obtain the risk information and send the client IA results. Thus, we also measured the following inter-process latency types: (1) pull: MRAAC Client obtains the current risk information from MRAAC Service via Android Binder. (2) broadcast: MRAAC Service broadcasts the stage updates to all MRAAC Client apps. (3) IA update: MRAAC Client sends its client IA results to MRAAC Service via Android Binder. Table 4 shows that the pull operation and the IA update had very short latency given that they both rely on Android Binder. The broadcast latency was acceptable given that in 95% of the experiments, the client app received the stage update from MRAAC Service within 20 ms on the Google Pixel phone.

The latency analysis results have shown that MRAAC necessitates extra processing time for multi-stage adaptation. However, when we factor in the reporting rate of contextual and behavioral events at the granularity of seconds, the time overhead of the multi-stage adaptation is negligible.

Battery consumption. We used the Battery Historian [22] to measure the battery consumption of the adaptation process of MRAAC Service on Google Pixel 3. We made the context detectors and the gait authenticator run at a fixed rate. We tested the following two settings and measured one-hour battery consumption five times for each setting: (1) Enabling adaptation. MRAAC was forced to perform one high-level adaptation (i.e., processing the `ON_SITE` and `OFF_SITE` signals) and one low-level adaptation (i.e., processing the `ON_FOOT_ENTER` and the `ON_FOOT_EXIT` signals) every 15 seconds (i.e., the maximum adaptation frequency). However, the gait authenticator was set to ignore the adaptation outcome and keep running. (2) Disabling adaptation. MRAAC did not process any adaptations. From the result, the average hourly consumption of the first setting was 2.95% and the second setting was 2.92%. It shows that the adaptation process introduced very low battery consumption to the device.

7.4 Use Case Simulation

We conducted a simulation based on the BYOD use case to show how MRAAC helps defend against unauthorized access with fewer false rejections. We used real-world sensor data to generate simulation traces and fed them into the MRAAC-integrated FairEmail app running on Google Pixel 3 in real-time so we could simulate unauthorized access and daily device usage events and log the adaptations, stage transitions, and authentication results of MRAAC. As introduced in Section 6.2, the multi-stage design enables us to implement an adaptive sliding window that adopts different (m, n) pairs based on the current stage. For simulation settings, we chose the majority vote as the final decision ($m = \lceil n/2 \rceil$) and selected three different n's: 5, 9, 11. A larger n targets a lower false rejection rate at the cost of longer reaction time and a higher false acceptance rate, which is suitable for lower-risk stages. Thus, for onsite stages, we set $n = 11$ for $D21$ and $n = 9$ for $D11, D22$; for offsite stages, we set $n = 9$ for $C21$ and $n = 5$ for $C11, C22$ (note: given the high risk of $C11$ and $C22$, we chose a small n to ensure low reaction time.) In addition to the adaptive sliding window, we adopted the touch authenticator for all stages except the locked stage and adopted the gait authenticator only at $C11$, $C21$, and $C22$.

For the simulation task, we randomly selected 10 users from the HMOG dataset [47], which includes motion sensor data and touch data of reading activities on a smartphone while walking or sitting. For each user, we used six sessions of data to train the authenticator models and used another two sessions (one "walking + reading" and one "sitting + reading") for simulation. We replaced raw sensor data with the HMOG data and matched the timestamp to real-world time. Since the HMOG dataset did not provide location data and app access data, we randomly generated location switch events (switching between onsite and offsite) and app switch events (switching between sensitive and non-sensitive resources). The interval between two switch events followed exponential distributions. We set the average intervals as one and two minutes for location switch

events and app switch events, respectively. We chose short intervals to trigger more adaptations to test the robustness of MRAAC.

Attack detection experiments. In the first set of experiments, we chose one user as the device owner and all the other users as attackers and repeated it for all users. We ensured the attackers' data was not used in the negative training data of the owner user. An attacker is supposed to access sensitive resources under the offsite context. We measured how long the adaptive authentication system took to lock out the attacker. We study two locking duration types: (1) t_h is the duration between the attacker picking up the device and the device locking out the attacker. (2) t_t is the duration between the attacker performing the first touch event and the device locking out the attacker. We study t_t separately, since the attacker might not operate on the device immediately after picking up the device. Among 90 attack events, MRAAC failed to detect only one attack when both the gait-based and touch-based authenticators falsely accepted the attacker. The average t_h was 33.2 s (95th percentile = 60.6 s). The average t_t was 11.0 s (95th percentile = 36.0 s), which implies that attackers did not have much time to launch an active attack before being blocked. Other biometrics-based IA mechanisms may be able to detect an attack even faster. Importantly, MRAAC is oblivious to the type of mechanism being used. Of course, without the use of an IA mechanism, the attack would not get detected at all. In summary, with MRAAC managing the authentication adaptation and performing multi-modal authentication, the authentication system can efficiently detect unauthorized access.

Usability experiments. A larger window increases the latency to detect an intruder but it reduces potential false positives. However, it is challenging to determine one single optimal window size for all circumstances. To address this problem, MRAAC's adaptive sliding window adapts the window size to the current context. It aims to reduce unnecessary authentication and mitigate false IA rejection, both of which can disrupt users and initiate EA.

In our second set of experiments, we compared MRAAC's adaptive sliding window approach with two constant-size sliding window schemes. To maintain a fair and consistent comparison, we used the maximum and minimum window values from the adaptive strategy to determine the sizes of our two baseline schemes: namely, the (3,5)-sliding window and the (6,11)-sliding window. During the experiments, we ran the complete trace of each user (the length of each trace was 10–15 minutes). We assume the user immediately passes EA and continues using the device if the system triggers EA. MRAAC performed 361 stage transitions for all 10 users without any undefined transitions. EA was triggered 15 times in total: 12 cases occurred at the access to sensitive resources and 11 cases occurred at the offsite stage (8 common cases when both conditions hold). Besides, no EA was triggered for 4 out of the 10 users. Figure 8 compares the raw authentication results of the gait and touch authenticators as well as two baseline methods to the aggregated results using the adaptive sliding window strategy enabled by MRAAC. The result shows that individual authenticators were prone to make false rejections due to their low model accuracy. Note that the gait authenticator produced fewer false rejection decisions because the user was not always walking. A larger window size ($n = 11$) can help reduce the false rejections of individual authenticators. However, MRAAC made the fewest false rejections among all methods for 9 out of 10 users. Only for user 998757, MRAAC activated EA once while the (6, 11)-sliding window did not activate any EA. It was triggered when MRAAC was adopting a small window size at a high-risk stage. We also note that the performance of MRAAC was bounded by the performance of individual authenticators. Nevertheless, we can see significant performance improvement by the multi-stage design of MRAAC. Then, we measured the activation times of the gait-based authenticators. Due to the adaptation mechanism, the gait authenticator was only active for 33.7 minutes out of 122 minutes for all experiments. The above simulation results have shown that MRAAC can help schedule authenticators to reduce unnecessary detection and false rejections.

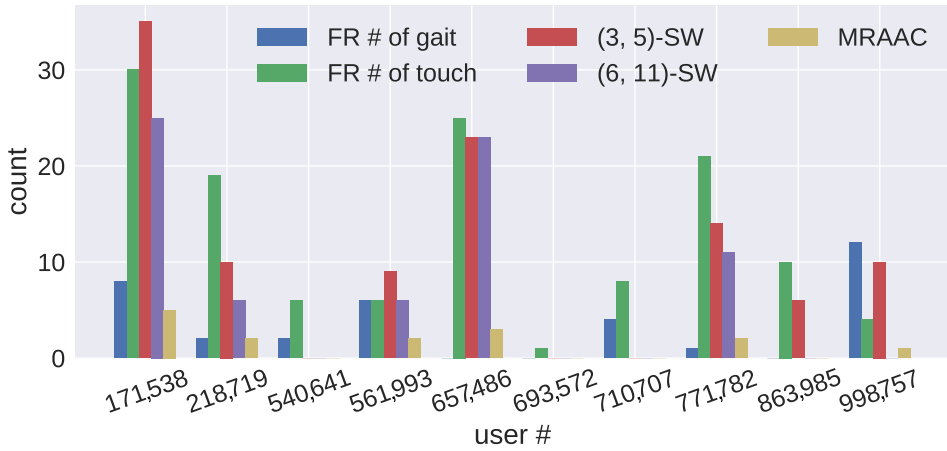

Fig. 8. The per-user raw false rejection numbers of authenticators and the EA numbers of baseline methods and MRAAC.

8 DISCUSSION

Client-server communications. MRAAC proposes a client-server structure for MDM/EMM solutions, which involves communications between the client app and the MRAAC service. Potential attacks may happen during the communication. On the one hand, the stage information broadcast to a client app may include or imply sensitive information (e.g., a user's location). Thus, the MRAAC service can only send digested or coarse-grained risk information that is essential to the client app. The client app should also declare the required permissions of sensitive risk information. On the other hand, the client app can send fake authentication results to MRAAC, which may lead to impersonation attacks (fake IA acceptance) and denial of service attacks (fake IA rejection). Thus, MRAAC only allows verified and trusted client apps to send authentication results (see Section 5.2.2) to avoid these attacks.

Accommodation of existing adaptive mechanisms. We propose MRAAC as a general adaptive authentication framework that is compatible with existing adaptive systems. Most existing adaptive authentication mechanisms [17, 40, 43] can be accommodated and abstracted as conditional policies. For example, CASA and PRISM use machine learning techniques to learn context models from sensor data and map the model output to a certain adaptation scheme that (de-)activates an authenticator or changes its parameters. Each model is a context provider and provides a signal for stage transition or an adaptation policy in MRAAC.

Modeling and policy-making. MRAAC is primarily provided for developers to design a multi-stage adaptive authentication system. Developers are responsible for designing use cases and using MRAAC to build the multi-stage model for their authentication system. However, end-users may have their own preferences and requirements of security and usability and may want to configure the adaptive authentication system. Existing studies [10, 11] have investigated how to enable users to customize authentication mechanisms. Thus, it should be possible for end-users to configure some components, such as authenticators and context providers (e.g., setting up a geofence for the trusted place [10]). End-users should also be able to determine what use cases to enable, but cannot modify the multi-stage model to change the adaptation flow of a certain use case. A possible avenue is to implement a usable configurable adaptive authentication system for end-users based on the MRAAC framework.

For deployment, all stages except the locked stage are transparent to end-users, i.e., stage transitions and adaptations are performed automatically and internally. End-users may perceive the existence of MRAAC only when EA is activated. Thus, the complexity of a multi-stage model is not a direct factor affecting end-users' normal device usage.

Usability study. Our simulation used traces generated from a real-world dataset to evaluate the usability of MRAAC in terms of the number of EA triggered. Existing studies [28] have covered the usability of IA mechanisms and shown that users are concerned with interrupt-authenticates (i.e., immediate device lock with EA). According to our experiment results, we can observe that MRAAC can significantly reduce the times of EA compared to individual IA mechanisms. However, performing a user study to collect user perceptions and feedback about MRAAC is future work.

9 CONCLUSION

We present a multi-stage risk-aware adaptive authentication and access control framework, MRAAC. It combines context and implicit authentication to dynamically adapt the authentication and access control mechanisms. The multi-stage design supports progressive and complex adaptation workflows. MRAAC provides two libraries to enable adaptive authentication and make third-party apps implement in-app control with risk awareness, respectively. Extensive experiments have shown the effectiveness of MRAAC in balancing security and usability of authentication systems with low performance overhead.

REFERENCES

[1] J. Abdella, Mustafa Özuysal, and Emrah Tomur. 2016. CA-ARBAC: Privacy preserving using context-aware role-based access control on Android permission system. *Secur. Commun. Netw.* 9, 18 (2016), 5977–5995. DOI: https://doi.org/10.1002/SEC.1750

[2] Shravan Aras, Chris Gniady, and Hari Venugopalan. 2019. MultiLock: Biometric-based graded authentication for mobile devices. In *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. 100–109.

[3] Patricia Arias-Cabarcos, Christian Krupitzer, and Christian Becker. 2019. A survey on adaptive authentication. *ACM Comput. Surv.* 52, 4 (2019), 1–30.

[4] Michael Backes, Sven Bugiel, Sebastian Gerling, and Philipp von Styp-Rekowsky. 2014. Android security framework: Extensible multi-layered access control on android. In *Proceedings of the 30th Annual Computer Security Applications Conference*. 46–55.

[5] Cheng Bo, Lan Zhang, Taeho Jung, Junze Han, Xiang-Yang Li, and Yu Wang. 2014. Continuous user identification via touch and movement behavioral biometrics. In *Proceedings of the IEEE 33rd International Performance Computing and Communications Conference (IPCCC'14)*. IEEE, 1–8.

[6] Marcel Bokhorst. 2021. FairEmail. Retrieved from https://github.com/M66B/FairEmail

[7] Ines Brosso, Alessandro La Neve, Graça Bressan, and Wilson Vicente Ruggiero. 2010. A continuous authentication system based on user behavior analysis. In *Proceedings of the International Conference on Availability, Reliability and Security*. IEEE, 380–385.

[8] Sven Bugiel, Stephen Heuser, and Ahmad-Reza Sadeghi. 2013. Flexible and fine-grained mandatory access control on Android for diverse security and privacy policies. In *Proceedings of the 22nd USENIX Security Symposium (USENIX Security'13)*. 131–146.

[9] Jiayi Chen, Urs Hengartner, and Hassan Khan. 2022. Sharing without scaring: Enabling smartphones to become aware of temporary sharing. In *Proceedings of the 18th Symposium on Usable Privacy and Security (SOUPS'22)*. USENIX Association, Boston, MA. Retrieved from https://www.usenix.org/conference/soups2022/presentation/chen

[10] Geumhwan Cho, Sungsu Kwag, Huh Jun Ho, Bedeuro Kim, Choong-Hoon Lee, and Hyoungshick Kim. 2021. Towards usable and secure location-based smartphone authentication. In *Proceedings of the 17th Symposium on Usable Privacy and Security (SOUPS'21)*.

[11] Alain Forget, Sonia Chiasson, and Robert Biddle. 2015. Choose your own authentication. In *Proceedings of the New Security Paradigms Workshop*. 1–15.

[12] Mario Frank, Ralf Biedert, Eugene Ma, Ivan Martinovic, and Dawn Song. 2012. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *IEEE Trans. Inf. Forens. Secur.* 8, 1 (2012), 136–148.

[13] Futureae. 2021. Futureae: Customer Stories. Retrieved from https://www.futurae.com/customer-stories/
[14] Lorena Gonzalez-Manzano, Jose M. De Fuentes, and Arturo Ribagorda. 2019. Leveraging user-related internet of things for continuous authentication: A survey. *ACM Comput. Surv.* 52, 3 (2019), 1–38.
[15] Aditi Gupta, Markus Miettinen, Nadarajah Asokan, and Marcin Nagy. 2012. Intuitive security policy configuration in mobile devices using context profiling. In *Proceedings of the International Conference on Privacy, Security, Risk and Trust and International Conference on Social Computing*. IEEE, 471–480.
[16] Dick Hardt. 2012. The OAuth 2.0 authorization framework. *RFC* 6749, Oct. (2012).
[17] Eiji Hayashi, Sauvik Das, Shahriyar Amini, Jason Hong, and Ian Oakley. 2013. CASA: Context-aware scalable authentication. In *Proceedings of the 9th Symposium on Usable Privacy and Security*. 1–10.
[18] Eiji Hayashi, Oriana Riva, Karin Strauss, A. J. Bernheim Brush, and Stuart Schechter. 2012. Goldilocks and the two mobile devices: Going beyond all-or-nothing access to a device's applications. In *Proceedings of the 8th Symposium on Usable Privacy and Security*. 1–11.
[19] Daniel Hintze, Matthias Füller, Sebastian Scholz, Rainhard D. Findling, Muhammad Muaaz, Philipp Kapfer, Eckhard Koch, and René Mayrhofer. 2019. CORMORANT: Ubiquitous risk-aware multi-modal biometric authentication across mobile devices. *Proc. ACM Interact., Mob., Wear. Ubiq. Technol.* 3, 3 (2019), 1–23.
[20] Google Inc. 2022. Android 10. Retrieved from https://source.android.com/security/enhancements/enhancements10
[21] Google Inc. 2021. Android Pin & Unpin Screens. Retrieved from https://support.google.com/android/answer/9455138?hl=en
[22] Google Inc. 2021. Battery Historian. Retrieved from https://github.com/google/battery-historian
[23] Google Inc. 2021. Measuring Biometric Unlock Security. Retrieved from https://source.android.com/security/biometric/measure
[24] Google Inc. 2021. Smart Lock. Retrieved from https://support.google.com/pixelphone/answer/6093922?hl=en
[25] Saad Inshi, Mahdi Elarbi, Rasel Chowdhury, Hakima Ould-Slimane, and Chamseddine Talhi. 2023. CAPEF: Context-aware policy enforcement framework for Android applications. *J. Eng. Res. & Sci.* 2, 1 (2023).
[26] Thomas Karanikiotis, Michail D. Papamichail, Kyriakos C. Chatzidimitriou, Napoleon-Christos I. Oikonomou, Andreas L. Symeonidis, and Sashi K. Saripalle. 2020. Continuous implicit authentication through touch traces modelling. In *Proceedings of the IEEE 20th International Conference on Software Quality, Reliability and Security (QRS'20)*. IEEE, 111–120.
[27] Hassan Khan, Aaron Atwater, and Urs Hengartner. 2014. Itus: An implicit authentication framework for Android. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*. 507–518.
[28] Hassan Khan, Urs Hengartner, and Daniel Vogel. 2015. Usability and security perceptions of implicit authentication: Convenient, secure, sometimes annoying. In *Proceedings of the 11th Symposium on Usable Privacy and Security (SOUPS'15)*. 225–239.
[29] Hassan Khan, Urs Hengartner, and Daniel Vogel. 2018. Augmented reality-based mimicry attacks on behaviour-based smartphone authentication. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. 41–53.
[30] Lukas Koller. 2021. Camera Roll Android App. Retrieved from https://github.com/kollerlukas/Camera-Roll-Android-App
[31] Masoud Mehrabi Koushki, Borke Obada-Obieh, Jun Ho Huh, and Konstantin Beznosov. 2021. On smartphone users' difficulty with understanding implicit authentication. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–14.
[32] Rajesh Kumar, Vir V. Phoha, and Abdul Serwadda. 2016. Continuous authentication of smartphone users by fusing typing, swiping, and phone movement patterns. In *Proceedings of the IEEE 8th International Conference on Biometrics Theory, Applications and Systems (BTAS'16)*. IEEE, 1–8.
[33] Ninghui Li, Qihua Wang, Wahbeh Qardaji, Elisa Bertino, Prathima Rao, Jorge Lobo, and Dan Lin. 2009. Access control policy combining: Theory meets practice. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*. 135–144.
[34] Shrirang Mare, Andrés Molina Markham, Cory Cornelius, Ronald Peterson, and David Kotz. 2014. Zebra: Zero-effort bilateral recurring authentication. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 705–720.
[35] Diogo Marques, Tiago Guerreiro, Luís Carriço, Ivan Beschastnikh, and Konstantin Beznosov. 2019. Vulnerability & blame: Making sense of unauthorized access to smartphones. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–13.
[36] Masoud Mehrabi Koushki, Borke Obada-Obieh, Jun Ho Huh, and Konstantin Beznosov. 2020. Is implicit authentication on smartphones really popular? On Android users' perception of "smart lock for Android." In *Proceedings of the 22nd International Conference on Human-computer Interaction with Mobile Devices and Services*. 1–17.
[37] Markus Miettinen, Stephan Heuser, Wiebke Kronz, Ahmad-Reza Sadeghi, and N. Asokan. 2014. ConXsense—Automated context classification for context-aware access control. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*. 293–304.

[38] Tempestt J. Neal, Damon L. Woodard, and Aaron D. Striegel. 2015. Mobile device application, Bluetooth, and Wi-Fi usage data as behavioral biometric traits. In *Proceedings of the IEEE 7th International Conference on Biometrics Theory, Applications and Systems (BTAS'15)*. IEEE, 1–6.

[39] Abena Primo, Vir V. Phoha, Rajesh Kumar, and Abdul Serwadda. 2014. Context-aware active authentication using smartphone accelerometer measurements. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 98–105.

[40] Arun Ramakrishnan, Jochen Tombal, Davy Preuveneers, and Yolande Berbers. 2015. PRISM: Policy-driven risk-based implicit locking for improving the security of mobile end-user devices. In *Proceedings of the 13th International Conference on Advances in Mobile Computing and Multimedia*. 365–374.

[41] Arun Kishore Ramakrishnan, Davy Preuveneers, and Yolande Berbers. 2013. A loosely coupled and distributed bayesian framework for multi-context recognition in dynamic ubiquitous environments. In *Proceedings of the IEEE 10th International Conference on Ubiquitous Intelligence and Computing and IEEE 10th International Conference on Autonomic and Trusted Computing*. IEEE, 270–277.

[42] Prathima Rao, Dan Lin, Elisa Bertino, Ninghui Li, and Jorge Lobo. 2011. Fine-grained integration of access control policies. *Comput. Secur.* 30, 2-3 (2011), 91–107.

[43] Oriana Riva, Chuan Qin, Karin Strauss, and Dimitrios Lymberopoulos. 2012. Progressive authentication: Deciding when to authenticate on mobile phones. In *Proceedings of the 21st USENIX Security Symposium (USENIX Security'12)*. 301–316.

[44] Felix Rohrer, Yuting Zhang, Lou Chitkushev, and Tanya Zlateva. 2013. DR BACA: Dynamic role based access control for Android. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC'13)*. ACM, 299–308. DOI: https://doi.org/10.1145/2523649.2523676

[45] Julian Seifert, Alexander De Luca, Bettina Conradi, and Heinrich Hussmann. 2010. TreasurePhone: Context-sensitive user data protection on mobile phones. In *Proceedings of the International Conference on Pervasive Computing*. Springer, 130–137.

[46] Babins Shrestha, Manar Mohamed, and Nitesh Saxena. 2019. ZEMFA: Zero-effort multi-factor authentication based on multi-modal gait biometrics. In *Proceedings of the 17th International Conference on Privacy, Security and Trust (PST'19)*. IEEE, 1–10.

[47] Zdeňka Sitová, Jaroslav Šeděnka, Qing Yang, Ge Peng, Gang Zhou, Paolo Gasti, and Kiran S. Balagani. 2015. HMOG: New behavioral biometric features for continuous authentication of smartphone users. *IEEE Trans. Inf. Forens. Secur.* 11, 5 (2015), 877–892.

[48] Max Smith-Creasey, Fatema A. Albalooshi, and Muttukrishnan Rajarajan. 2018. Context awareness for improved continuous face authentication on mobile devices. In *Proceedings of the IEEE 16th International Conference on Dependable, Autonomic and Secure Computing, 16th International Conference on Pervasive Intelligence and Computing, 4th International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/-DataCom/CyberSciTech'18)*. IEEE, 644–652.

[49] Adam Wójtowicz and Krzysztof Joachimiak. 2016. Model for adaptable context-based biometric authentication for mobile devices. *Person. Ubiq. Comput.* 20, 2 (2016), 195–207.

[50] WSO2. 2021. WSO2: Strong Authentication. Retrieved from https://wso2.com/identity-and-access-management/strong-authentication/

[51] Yunze Zeng, Amit Pande, Jindan Zhu, and Prasant Mohapatra. 2017. WearIA: Wearable device implicit authentication based on activity information. In *Proceedings of the IEEE 18th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'17)*. IEEE, 1–9.

[52] Qin Zou, Yanling Wang, Qian Wang, Yi Zhao, and Qingquan Li. 2020. Deep learning-based gait recognition using smartphones in the wild. *IEEE Trans. Inf. Forens. Secur.* 15 (2020), 3197–3212.

Received 24 June 2022; revised 28 December 2023; accepted 1 February 2024