On the Privacy Risks of Compromised Trigger-Action Platforms

Yu-Hsi Chiang[1(✉)], Hsu-Chun Hsiao[1], Chia-Mu Yu[2], and Tiffany Hyun-Jin Kim[3]

[1] National Taiwan University, Taipei, Taiwan
`r06922023@csie.ntu.edu.tw`
[2] National Chiao Tung University, Hsinchu, Taiwan
[3] HRL Laboratories LLC, Malibu, CA, USA

Abstract. Trigger-action platforms empower users to interconnect various physical devices and online services with custom automation. While providing convenience, their centralized design raises privacy concerns for end users. Unlike prior work that consider privacy leakage to action services, we consider privacy leakage to compromised platforms. After investigating potential privacy exposure to a popular trigger-action platform, IFTTT, we identified three types of leakages: *event data, trigger event presence, and device possession*. We also found that 91% of the top 500 triggers on IFTTT potentially leak sensitive information to the platform, and 25% leak implicitly. To achieve the paradoxical goal of hiding the event data and presence while asking the platform to trigger corresponding actions when an event occurs, we propose Obfuscated Trigger-Action Platform (OTAP) and Anonymous Trigger-Action Platform (ATAP). ATAP additionally provides *device set confidentiality* at the cost of minor platform modification. Our schemes can preserve user privacy without sacrificing convenience, and are incrementally deployable in various use cases. Our work addresses a crucial missing piece in securing the trigger-action ecosystem, and can be integrated with solutions that ensure integrity against untrusted platforms or solutions that address untrusted vendor services and users.

1 Introduction

Fueled by the growing demand for home automation, *trigger-action platforms* (e.g.., IFTTT [7], Zapier [11], and Microsoft Power Automate [1]) are gaining popularity among smart home users. Such cloud-based platforms empower users to connect heterogeneous devices from different vendors and write automation rules, which often take the form of "If a certain *trigger* occurs, then do a certain *action*."[1] In addition to home devices, these platforms are also integrated with many online services such as Google, Amazon, and Instagram. Due to their ease of use and the variety of supported services, these platforms have attracted millions of users running billions of rules per month [8].

[1] For example, "If I am approaching home, turn on the air conditioner.".

L. Chen et al. (Eds.): ESORICS 2020, LNCS 12309, pp. 251–271, 2020.
https://doi.org/10.1007/978-3-030-59013-0_13

The convenience comes at the cost of privacy. Because trigger-action platforms are authorized to collect and manipulate data from various services, some of which may contain sensitive information including users' location, purchase history, private social media posts or photos, privacy concerns are raised if the platforms are untrusted or hacked. Moreover, cloud-based centralized architecture makes these platforms an attractive target for attackers, as demonstrated by recent incidents targeting the cloud services of famous companies, including Equifax [9], Yahoo [3,4] and Target [2]. Therefore, unlike prior work [13,14,23,28] which focused on the privacy leakage to action services or rule authors through improperly configured rules, we consider the case of compromised platforms. We also note that a compromised trigger-action platform is more dangerous than any single compromised vendor (e.g., Samsung, Google, etc.) due to the amount of aggregated information.

To better understand the privacy risks of a compromised trigger-action platform, we first conduct a case study on one popular trigger-action platform, IFTTT. Instead of categorizing triggers by the functionality of their associated services [13], we categorize by the types of information they may leak, and identified three types of leakage: *event data, trigger event presence, or device possession.* Among the most popular 500 triggers, we found that 91% potentially send sensitive data to IFTTT, and 25% may leak sensitive information implicitly by the presence of their triggers. For example, a hearing-aid device leaks health conditions, and a surfing sports application leaks a user's personal interest. Categorizing potentially-leaking information types revealed that IFTTT can build a profile of a user who installs a sufficient number of rules.

To enhance the privacy of the trigger-action ecosystem in a practical manner, we investigate the problem of preventing an untrusted platform from learning users' private information (via either *received data, trigger event presence, or device possession*), while simultaneously keeping the benefits of a cloud-based solution. The goal seems paradoxical, as we would like to maintain both of the trigger data and trigger event presence secrecy, while asking the platform to trigger corresponding actions when an event occurs. Existing solutions based on encryption or fine-grained access control such as DTAP [20] fail to achieve this goal, as they can only hide the data but not the presence of triggers, and simply limiting access to sensitive information could affect the platforms functionality. In this work, we demonstrate that it is possible to build such schemes using *obfuscation* and *anonymization* techniques, and propose (1) Obfuscated Trigger-Action Platform (OTAP) and (2) Anonymous Trigger-Action Platform (ATAP). In short, OTAP hides real triggers by also sending fake triggers to confuse the platform, and ATAP preserves users' privacy by breaking the links between a user and his/her data. According to our empirical evaluation using realistic configurations, our schemes can balance the trade-offs between security and utility under different assumptions and requirements. For example, our schemes add at most 25 ms to rule execution time and can achieve the same throughput as existing solutions with at most 8× computation resources.

To summarize, this work makes the following contributions:

- We conducted an empirical study on potential privacy exposures to trigger-action platforms. We categorized the types of leaked information and the ways they are leaked.
- We designed and implemented two privacy-preserving trigger-action platform solutions, OTAP and ATAP, based on trigger *obfuscation* and *anonymization*. OTAP hides information about data and trigger event presence. ATAP additionally hides information about device/account possession.
- Through security analysis and empirical experiments, we show that our proposed schemes can be deployed in various practical scenarios.

2 Problem Definition

Our core goals are to (1) identify the privacy risks of trigger-action platforms and information leakage types, and (2) alleviate the exposed privacy risks in a practical manner. We provide formal privacy framework and proofs in Appendix A.

2.1 Threat Model

We consider an untrusted trigger-action platform that can be compromised and violate end-users' confidentiality. An adversary can collect users' information through the following three sources:

Data & parameter: The raw data of trigger and action parameters and trigger event data.

Trigger event presence: The occurrence of trigger events. For example, the occurrence of trigger-event "leave home" allows an attacker to infer if the user is at home.[2]

Device/account possession: Status of users' ownership of devices or online accounts. For example, if one connects a hearing aid (e.g.., Oticon [6]), an attacker may infer that the user has hearing issues.

We do not consider action presence in our threat model since no information is sent back from the action service to the platform when an action is fired. Thus, there is no way for the platform to verify whether an action is indeed executed. We also assume that the platform will not collude with partner services and there are secured channels (e.g., HTTPS) between different parties.

Since we only focus on confidentiality in this work, the following attacks are considered out of our scope.

- Breaking action integrity, including replaying old triggers or triggering unauthorized actions, which can be mitigated with existing solution [20].
- Denial-of-service, which has low incentives from platforms' perspectives as DoS attacks will drive users away and hurt the platform's reputations.
- Leaking sensitive information to partner services.

[2] Turn off WiFi on your Android when you leave home to save power [10].

2.2 Desired Properties

In respect to Sect. 2.1, we define the following three types of confidentiality that a privacy-preserving trigger-action platform should preserve.

Table 1. Example of a trigger-action rule

Rule	If saying a specific phrase to Google Home, then set the Hue light brightness
Trigger	Say a phrase with a number
Trigger params	phrase: `set hue brightness to #`
Action	Set brightness
Action params	brightness: *number_spoken*
Event data	number_spoken: 50 triggered_at: 2020/01/01 12:00

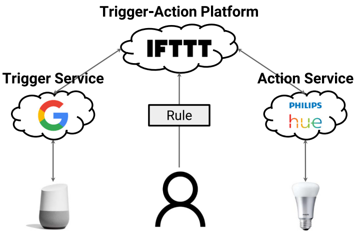


Fig. 1. Example of a trigger-action ecosystem

Data & Parameters Confidentiality. The platform should not learn the trigger and action parameters and the event data.

Trigger Event Presence Confidentiality. The trigger-action platform should not be capable of infering when or whether a trigger event occurs.

Device Set Confidentiality. The trigger-action platform should not learn what devices or service accounts a user possesses.

For practicality, the following should also be taken into account.

Minimal Information Exposure. Every entity in the system should not learn additional information compared to what it knows from the original scheme (i.e., IFTTT). Otherwise, a solution simply delegates the job of the platform (i.e., rule storage and execution) to partner services, which does not protect users' privacy. In the existing IFTTT ecosystem, the trigger and action services know the trigger and action-related information, respectively, while the platform knows both of the information and the rules. Though the attacker we considered in this work is the platform, we would also want to ensure that the proposed solutions do not leak additional information to either of the trigger or action service so that the privacy risks are not shifted or raised.

Performance. The system should be able to handle thousands of concurrent triggers per second, the rule execution should be fast enough, and the delay between the trigger and action being done should be reasonable (e.g.., 15 min of maximum delay in IFTTT as of 2020).

3 Case Study: Privacy-Risks of IFTTT

To understand the security risks of a compromised trigger-action platform, we inspect the trigger specifications of 500 most popular triggers on IFTTT, and investigate the types of information that IFTTT can learn and the sources of leakage. Datasets are available [12].

3.1 Background: IFTTT & Trigger-Action Platforms

A trigger-action platform is a cloud service that allows end-users to connect different IoT devices and online services based on conditional *rules*. The rules are in the form of "IF a certain trigger happens, THEN do a certain action," such as "IF saying a specific phrase to Google Home, THEN set the Hue light brightness," or "IF someone posts a new photo on Instagram, THEN upload it to my Google Drive." As shown in Table 1, triggers and actions are provided by certain third-party partner *services* such as Phillips Hue, Instagram, or Google, and they are customized by the trigger and action parameters, respectively.

A rule will be triggered when there is an *event* that satisfies the condition described by its trigger and trigger parameters. *Event data* represents the data associated with the event, and the event data can be used as part of action parameters. Some platforms also allow users to write a piece of code (called *filter code*), which filters and transforms the event data.

Figure 1 illustrates a typical ecosystem of trigger-action platforms [27]: upon executing a rule, the trigger-action platform receives the data from the trigger service, transforms them into the parameters of the action, and calls the API of the action service. The platform usually does not connect to devices directly, but through cloud services provided by their vendors for ease of management. Partner services communicate to their devices using proprietary protocols, and use the platform's web-based protocols to fire triggers or receive action requests.

3.2 Methodology

Dataset. We obtained the list of event-data fields (known as ingredients in IFTTT) of each trigger using IFTTT's undocumented GraphQL API [12]. We queried in March 2019, and retrieved all of the 524 services and 2,396 triggers. We identified the 500 most popular triggers based on a popularity rank derived from a public dataset collected in May 2017 [27]. Their dataset includes about 280,000 rules of IFTTT, and contains the number of installed users for each rule. We assessed the popularity of each trigger by summing up the number of installed users having that trigger.

Approach. We assigned several labels to each trigger to indicate what types of information may be leaked and how the leakage would occur. To indicate the possibility of containing private information, we assigned a score from 1 to 3: Scores of 1, 2, or 3 indicate that the data does not contain, possibly contains, or absolutely contains private information, respectively. For example, a "photo uploaded" trigger may leak private information depending on the content of the photo and hence we assigned the score of 2; an "arrive at" trigger definitely leaks a user's location and hence we assigned 3.

3.3 Results

Types of Leaked Information. As shown in Table 2, we identified 12 categories of leaked information, and labeled triggers for each category. Note that a

Table 2. Categories of leaked information

Basic information	Name, age, email address, phone number, physical address, occupation, etc.
Location	Current location, location history
Finance	Bank account balance, transaction logs, purchase history, etc.
Health status	Weight, BMI, blood pressure, heart rate, etc.
Social	Contacts, membership status of organizations, etc.
Communication	Call records, SMS, voicemail, email, social media posts, etc.
Lifestyle/Activities	Data or logs that show user's activities. e.g., the time the user wakes up, turns on the TV or goes to work, etc.
Activities summary	The summary of user's activities. e.g., total working hours, distance of driving, etc.
Interest	Videos liked or watched, interested news topics, followed stocks, etc.
Physical sensor	Temperature of a room, indoor CO2 level, etc.
Others (Personal)	Information not publicly available and not fit in any of the above categories
Public	Publicly available information. e.g., weather forecasts, Public posts, etc.

Table 3. Sources of leakage ($n = 500$)

Sources of leakage	# of triggers	%
Event data	456	91%
Trigger event presence	127	25%
Device/account possession	2	0.4%

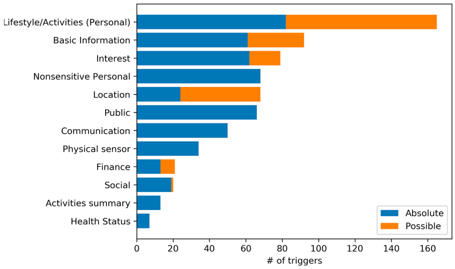


Fig. 2. # of triggers for each category

trigger may be assigned to more than one category. This list captures the most common types, but is not exhaustive.

Among these triggers, 91% leak at least one type of private information (i.e., not labeled as public). As shown in Fig. 2, lifestyle/activities category is a broad category into which most of the state change events fall, and hence is most likely to be leaked, while health status is least likely.

Sources of Leakage. We labeled the sources of leakage of each trigger event based on our definitions in Sect. 2.1. We found that a trigger may leak information through multiple sources, and triggers that leak information by trigger event presence or device/account possession will always leak information by data too.

Table 3 shows the number of triggers subject to each of the three leakage sources. We count the number of triggers assigned a score 2 or 3. We found the most of the information is leaked through event data. Even if event data were to be hidden from IFTTT, 25% of triggers still leak information by their presence.

4 Proposed Solutions

We propose OTAP and ATAP that achieve different sets of security goals as outlined in the previous section. Table 4 provides a preview of comparisons of the different schemes. Besides IFTTT, we also compare our schemes with a Pairwise Connection approach, in which trigger and action services connect to each other directly without communicating with the platform.

Pairwise Connection achieves the security goals, since no information is given to the platform. However, Pairwise Connection breaks *minimal information exposure*, as a trigger service can gain the information of what other device a user owns if it connects to the action service. Also, removing the platform from the system makes the maintenance more difficult. Without a trigger-action platform, a service must notify all other services when it changes its API, but with a trigger-action platform, the service only needs to notify the platform.

Table 4. Comparison of Schemes. Properties 4–6 are compared against the baseline. The Pairwise Connection approach has no platform, and thus trivially achieves Properties 1–3 and 5 (*).

Property/Scheme	IFTTT (baseline)	Pairwise connection	OTAP	ATAP
1. Data & Parameters Confidentiality	No	Yes*	Yes	Yes
2. Trigger Event Presence Confidentiality	No	Yes*	Yes	Yes
3. Device Set Confidentiality	No	Yes*	No	Yes
4. Minimal Information Exposure	-	No	Yes	Yes
5. No Platform Modification Required	-	Yes*	Yes	No
6. No Service Modification Required	-	No	No	No
7. Ease of Maintenance	Easy	Hard	Easy	Easy

Core Techniques. In both OTAP and ATAP, encryption is employed to provide *data & parameters confidentiality*. One challenge is that, due to the requirement of *minimal information exposure*, the trigger and action services should not connect to each other directly, thus making key establishment a non-trivial task.

Another challenge is to provide *trigger event presence confidentiality*. Indeed, it sounds paradoxical to hide triggers from the platform while asking the platform to execute certain rules when a trigger is fired. To that end, we propose two techniques: *trigger obfuscation* and *trigger anonymization*.

For trigger obfuscation, we can notify the platform when some trigger occurs, along with fake triggers when there is no actual event. In this way, the platform cannot be sure whether there is a real event happening when receiving triggers.

On the other hand, the idea of trigger anonymization is different. We still allow the trigger-action platform to learn when a trigger occurs, but make it impossible to associate the trigger with a certain user. The advantage of this technique is that, in our system, we already have trigger and action services which can act as proxies or "mixes" naturally, and thus it is unnecessary to add other parties for anonymization.

For the rest of this paper, we assume that user U wants to setup a rule r, the associated trigger and action services are T and A, respectively, and the trigger-action platform is denoted by P.

4.1 Obfuscated Trigger-Action Platform (OTAP)

OTAP is based on trigger *obfuscation* and is designed to protect both *data & parameters confidentiality* and *trigger event presence confidentiality* without platform modification. The protocol flow of OTAP is the same as the existing IFTTT flow except that encryption is used to provide *data & parameters confidentiality.* In addition, to obfuscate the trigger-action platform, the trigger service periodically sends out a trigger event, either real or fake, every τ seconds, where τ refers to *trigger period* and the time points are called *trigger points.* Note that τ is configurable by the user and will be part of the trigger parameters. A symmetric encryption is used and the encryption function is denoted by $\mathsf{Enc}(k, m)$, where k is the encryption key and m is the message to be encrypted. We also introduce two types of encryption keys:

- **User-Service Keys**: For each user U and service S, there is a shared key k_{US}, which will be used for encrypting trigger and action parameters.
- **Rule-Specific Keys**: For each rule r, there is a key k_r shared by the corresponding trigger and action services, and is used for encrypting event data.

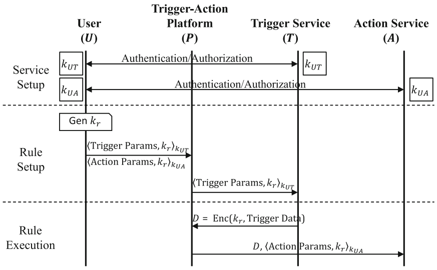

Trigger-Action Platform (P)
User (U)
Trigger Service (T)
Action Service (A)
Service Setup
k_{UT}
Authentication/Authorization
k_{UT}
k_{UA}
Authentication/Authorization
k_{UA}
Gen k_r
Rule Setup
$\langle$Trigger Params, $k_r\rangle_{k_{UT}}$
$\langle$Action Params, $k_r\rangle_{k_{UA}}$
$\langle$Trigger Params, $k_r\rangle_{k_{UT}}$
Rule Execution
D = Enc(k_r, Trigger Data)
D, $\langle$Action Params, $k_r\rangle_{k_{UA}}$

Fig. 3. OTAP protocol

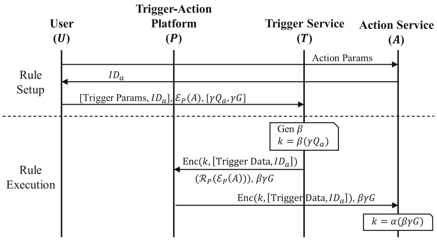

Trigger-Action Platform (P)
User (U)
Trigger Service (T)
Action Service (A)
Rule Setup
Action Params
ID_a
[Trigger Params, ID_a], $\mathcal{E}_P(A)$, [γQ_a, γG]
Gen β
$k = \beta(\gamma Q_a)$
Rule Execution
Enc(k, [Trigger Data, ID_a])
($\mathcal{R}_P(\mathcal{E}_P(A))$), $\beta\gamma G$
Enc(k, [Trigger Data, ID_a]), $\beta\gamma G$
$k = \alpha(\beta\gamma G)$

Fig. 4. ATAP protocol

We now present the phases of the protocol, as shown in Fig. 3. From the user's perspectives, note that the key managements and encryption/decryption can be done by installing a local app or a browser plugin to ease the burden.

Phase 1: Service Setup. Besides the authentication and authorization flows in the original scheme, in OTAP, the user's client (e.g.., local app or browser plugin) establishes the shared keys k_{UT} and k_{UA} with T and A during this phase.

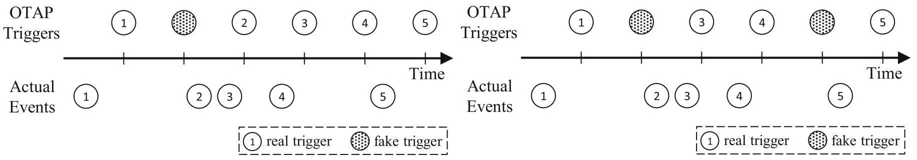
Phase 2: Rule Setup. Before uploading the rule to the platform, the rule-specific key k_r must be obtained. As highlighted earlier, the trigger and action services should not be directly connected. Thus, OTAP relies on the user's client to do this job since the user can connect with both parties. After generating k_r, the key will be uploaded by the user client to the platform as part of the rule along with the trigger and action parameters. If there are multiple actions in a single rule, then all of the involved action services will share the same key.

To ensure security and *parameter confidentiality*, the parameters must be encrypted, Therefore, the actual messages received by the platform will be $\langle \text{Trigger Params}, k_r \rangle_{k_{UT}}$ and $\langle \text{Action Params}, k_r \rangle_{k_{UA}}$, where

$$\langle m, k_r \rangle_k = [\mathsf{Enc}(k, m), \mathsf{Enc}(k, k_r), \mathsf{HMAC}(k, \mathsf{Enc}(k, m) \| \mathsf{Enc}(k, k_r))].$$

The first two terms are the parameters and k_r encrypted by the user-service key, and the last term is a Message Authentication Code providing integrity check.

Phase 3: Rule Execution. The trigger service is able to decrypt the trigger parameters and rule-specific key k_r using k_{UT}. At each trigger point, T sends to the platform a trigger event, with event data encrypted by k_r. The platform then forwards the encrypted data, encrypted action parameters, and the key to the action service. Upon receiving these, the action service first decrypts using k_{UA} to obtain the action parameters and k_r. Finally, with k_r, the action service decrypts event data and perform the action.



(a) Guaranteed Delivery

(b) Latest-Only Delivery

Fig. 5. Illustration of two delivery strategies

Note that the size of the encrypted event data can still leak sensitive information. However, we observed that the trigger fields of the same type are in fixed size in most cases. For example, the temperature is always an integer with the size of one block after encryption. If such a side-channel attack is a concern, the trigger service can pad the trigger field to its maximum allowed size.

To summarize, in OTAP, the information learned by the trigger and action services is the same as the existing solution. The platform knows the rules (i.e., which trigger and action are related), but not trigger and action data as they are encrypted. Trigger presence is also hidden by having fake triggers. The delay, whether deterministic or not, does not expose information about the real triggered events, as real triggers are indistinguishable from fake ones in OTAP.

Obfuscated Trigger and Delivery Strategies. Thus far, we have claimed that a fake event may be sent out at some trigger points. However, we have not discussed how to decide when to send out a fake trigger and how to cancel a false action when a fake trigger is sent. The latter issue can be caused because the platform is unaware of the truthfulness of the trigger and fires an action even if the incoming trigger is fake. To address this issue, we add an additional Boolean field is_real to the event data so that the action service can safely discard the message when it is set to be $false$. Since the flag resides in the event data, which is encrypted, the platform is unable to infer its value.

As for the former issue, we propose two main strategies called *guaranteed delivery* and *latest-only delivery* to decide which trigger is to be sent at each trigger point, as illustrated in Fig. 5. The guaranteed delivery strategy ensures that every real trigger event is sent out. Specifically, the trigger service will queue all the trigger events and send out the oldest unreported event at trigger points. However, the delay (the time between when an event occurs and is fired) may increase quickly if the trigger period is not short enough.

In contrast, the latest-only delivery strategy keeps only the latest trigger that is not yet fired. Hence, if many trigger events occur between two consecutive trigger points, only the latest one would be sent out. This strategy effectively shortens the delay, but may lead to event loss (e.g., event 2 in Fig. 5b).

How to choose the best trigger period and delivering strategy depends on the type of trigger and the rule, and can be adjusted dynamically to adapt the scenario of use. For example, for the rule "If the motion sensor detects I'm in the room, then turn on the light," one might want to have a short trigger period so that the action can be fired near real-time.

As for the delivery strategies, the guaranteed delivery is probably a better option for rules like "Log the time I enter and leave the office and home," as one would simply like all the events to be logged and might care less about delays. For the rule "If the temperature changes, adjust the thermostat," one may choose the latest-only delivery strategy as only the most recent temperature is the relevant information.

Finally, one may also wonder that if it is possible to use a randomized trigger period instead of a constant τ. Indeed, using a randomized trigger period will not weaken the security as long as the distribution of the trigger period is independent of actual trigger events. However, as shown in Appendix B, among all trigger period distributions of the same mean value, the constant trigger period will minimize the average delay. Note that the selection of trigger period and strategy will only affect the utility but not security.

To integrate OTAP with the existing IFTTT ecosystem, the user clients (i.e., app or browser plugin) and partner services need to implement the encryption and decryption functionalities, while *no platform modification* is required. The existing IFTTT basically sends the user-provided parameters to the partner services and forwards the data from the trigger service to the action service. In OTAP, the only difference is that the parameters and data are now encrypted and an additional parameter is added (i.e., the rule-specific key). Since the platform can operate without knowing the actual content of parameters and data, there is no need to change the platform's implementation.

4.2 Anonymous Trigger-Action Platform (ATAP)

ATAP achieves all three security goals at the cost of trigger-action platform modification. The scheme preserves trigger event presence confidentiality based on the idea of anonymization. The trigger service will drop user information before sending out a trigger to the trigger-action platform.

However, with this approach, we can no longer rely on the trigger-action platform to execute the rule. This is because if the platform were to know which rule to execute, it can distinguish between triggers of different users, contradicting our goal. Therefore, we have to store the rules on trigger and action services and use the platform only for forwarding, which also achieves our intention to preserve *device set confidentiality.* Although most of the functionality of the platform are discarded, we achieve *minimal information exposure* and easier system maintenance by keeping the platform, which the Pairwise Connection approach of removing the platform fails to do so. Also, as the approach is based on anonymization, its security is bounded by the number of users of the system.

Before diving into the protocol, we first introduce some cryptographic-related notations and techniques that will be used. We use a symmetric encryption function denoted by $\mathsf{Enc}(k, m)$, where k is the key and m is the plaintext to be encrypted. We use $\mathcal{E}_S(m)$ to represent the ciphertext of m encrypted by S's public key under ElGamal. Finally, an elliptic curve $\mathcal{G}$ is also used in ATAP with G being its base point.

To preserve data confidentiality, we need to encrypt the event data and a shared key must be established between the trigger and the action services. OTAP accomplishes this requirement by having the user's client generate the key and distribute it to both services. However, ATAP requires different approach because the triggers are anonymized. Specifically, the action service cannot determine which key to use for decryption upon receiving the event data. Public key exchange protocols, such as ECDH, would break *minimal information exposure*, since the trigger and action service can identify their counterparts by matching the public key. Therefore, the public key needs to be randomized. The modified ECDH protocol is described as follows.

Suppose that the ECDH public key of A is Q_A and the associated private key is α (i.e., $Q_A = \alpha G$). The protocol consists of three steps:

1. Client U selects a random scalar γ and sends γQ_A, γG to trigger service T.
2. T selects another random scalar β and sends $\beta\gamma G$ to action service A.
3. T and A can both compute the shared key $k = \beta(\gamma Q_A) = \alpha(\beta\gamma G)$.

Figure 4 shows the protocol of ATAP. Note that there is no service setup phase since the user no longer needs to register on the trigger-action platform.

Phase 1: Rule Registration. When setting up a new rule, the client starts by registering the action with A and receives an action ID ID_a. Then the client registers the trigger with T. The information given to T falls into three groups.

1. Trigger params and ID_a: Information needed for trigger-action.
2. $\mathcal{E}_P(A)$: The name of the action service encrypted by P's public key, such that the platform knows where the event data should be forwarded to.
3. γQ_A and γG: Used for ECDH key exchange, as described previously.

Phase 2: Rule Execution. When a trigger event occurs, the trigger service simply fires the event to the platform. The event data and ID_a will be encrypted by the shared key k. Similar to OTAP, the event data can be padded to prevent leaking sensitive information through data sizes, if necessary. The service name is re-randomized through $\mathcal{R}_P(\mathcal{E}_P(A)) = \mathcal{E}_P(A) + \mathcal{E}_P(0)$ exploiting the homomorphic property of ElGamal. The key exchange message $\beta\gamma G$ is also sent out. Upon receiving the messages, the platform obtains the service name A by decrypting $\mathcal{R}_P(\mathcal{E}_P(A))$, and forwards the rest of the messages to A. The action service computes the shared key k and decrypts the event data and action ID ID_a. With ID_a, the action service can find the corresponding action parameters and fire the action.

Re-randomization is needed for $\mathcal{E}_P(A)$ since $\mathcal{E}_P(A)$ is fixed for a given rule. Without re-randomization, some information can be used to fingerprint the user—the platform can learn that two different triggers are from the same user. Re-randomization is required to limit the attack impact: even if the attacker can break trigger presence confidentiality of one trigger, he/she cannot easily break trigger presence confidentiality of other triggers by linking them together.

To sum up, in ATAP, the trigger and action services know only the information of trigger and action, respectively. The platform knows nothing but some triggers of a service occurred and some actions of a service were fired.

4.3 Security Analysis

We provide security proof sketc.hes here and detailed analysis in Appendix A.

Assume the use of IND-CPA symmetric encryption scheme $\mathsf{Enc}(\cdot)$ (e.g., AES-CBC) and IND-CPA asymmetric encryption scheme $\mathcal{E}(\cdot)$ (e.g., ElGamal encryption). For OTAP, since all transmitted messages are encrypted by $\mathsf{Enc}(\cdot)$, data & parameter confidentiality is satisfied. By the definition of IND-CPA, an adversary cannot distinguish between pairs of ciphertexts even with the knowledge of plaintexts, and hence an adversary cannot distinguish between real and fake trigger events. Hence, OTAP preserves trigger event presence confidentiality.

For ATAP, all transmitted messages are encrypted by either $\mathsf{Enc}(\cdot)$ or $\mathcal{E}(\cdot)$, and thus data & parameters confidentiality is satisfied. In addition, since IND-CPA leads to randomized encryption, this implies that the adversary in ATAP is unable to link between different triggers, achieving trigger event presence confidentiality. Device set confidentiality is also achieved, as no further information is provided to the platform.

5 Performance Evaluation

To evaluate the performance of our proposed schemes, we implemented prototypes of our systems and a skeleton version of IFTTT as our baseline (referred to as the plain scheme). We ran several experiments to measure the end-to-end latency and the throughput of our systems, as described in Sects. 5.1 and 5.2.

Experiments Settings. We implemented all the partner services and trigger-action platforms as web servers using Node.js, and used MongoDB as the back-end database. We used 128-bits AES-GCM for symmetric encryption and curve P-256 for elliptic curve algorithms. Inspired by Fernandes et al. [20], we use the rule “IF new item added to TODO list contains the word ‘cat’, THEN send an SMS of this item.” This contains all the essential elements of a typical rule: a trigger parameter that indicates the trigger condition, and using event data as action parameters.

We hosted each service on separate Google Compute Engine g1-small instances. Each instance was configured with 1 Intel Xeon vCPU @ 2.2 GHz, 1.7 GB memory, and 10 GB SSD storage in Ubuntu 18.04.

5.1 Latency

We compare end-to-end latency of the privacy-preserving versions with the plain version. The latency increment mainly derived from (1) computational overhead of the cryptographic operations, and (2) the delay caused by sending triggers periodically, only for OTAP.

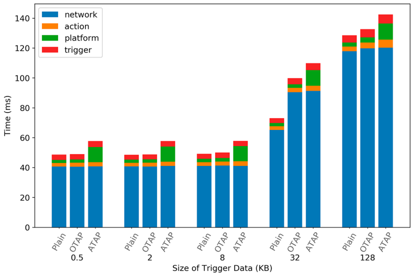


Fig. 6. Latency breakdown

Table 5. Goodput of different schemes

Scheme	Avg. (req/sec)
Plain	411.78
OTAP ($\tau = 17$, guaranteed)	52.46
OTAP ($\tau = 1796$, latest-only)	162.64
ATAP	69.54
ATAP (w/o re-randomization)	360.27

Computational Overhead. We measured the time spent at different services and the network overheads. The end-to-end latencies are largely affected by the size of the event data, as they need to be encrypted and decrypted by the trigger and the action services, respectively. Therefore, we varied their size from 0.5 KB to 128 KB, and the experiment result is shown in Fig. 6.

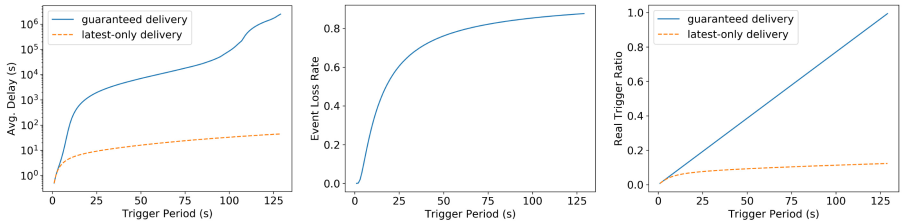
A gap exists between the network overheads of the plain scheme and OTAP (ATAP) when the trigger size is 32 KB, which is caused by TCP congestion control. With encryption, transferred data is slightly bigger than the congestion window size, requiring an additional round-trip for OTAP and ATAP. Despite network overhead, the results show that OTAP adds at most 5 ms latency while ATAP adds at most 25 ms. Both are small compared to network overheads or other delays, given that there are no real-time requirements in these systems.

Triggering Delay. For OTAP, we conducted an empirical study on how the trigger period affects the delay between the time a trigger occurs and the time it is sent out. We used the CASAS hh104 dataset [17], which contains sensor data that was collected in the home of a volunteer adult for about two years. We selected the MA022 motion sensor events as triggers, which are the most frequent events in this dataset. Appendix C describes the event distribution.

We ran the experiment for both delivery strategies, and the result is in Fig. 7a. For guaranteed delivery, the delay grows quickly as the trigger period increased; in this case, if we still want to guarantee real-time triggering, then the trigger period should be set around the bursty rate of the trigger (2 s in this example). For latest-only delivery, in contrast, the delay remains low as the trigger period increases; the delay is around one-third of the trigger period. However, there is a trade-off between the delay time and trigger event loss. For reference, we show the resulting event loss rate of different trigger periods in Fig. 7b.

5.2 Goodput

Besides the end-to-end delay of a single rule execution, we also examine the number of rules our schemes can handle, which affects the delay. We define the goodput to be the maximum number of *real* rules being executed per second. The rule we used in this experiment is the same as in Sect. 5.1 with size being 0.5 KB, and we used ApacheBench [5] to conduct the experiment by sending 10,000 trigger activations with up to 400 concurrent connections. The results are presented in Table 5. We set the concurrency level to 400 by estimating the maximum concurrent execution of the current IFTTT. According to their website [8], around 1 billion rules are executed per month, which means, on average, approximately 385 rules are executed per second.



(a) (b) (c)

Fig. 7. Trigger period vs. (a) Delay, (b) Event loss rate, (c) Real trigger ratio

For OTAP, the goodput depends heavily on the delivery strategy and the trigger period τ. A longer trigger period will not only increase the goodput but also increase the latency, as shown in Figs. 7a and 7c. In the experiment, we choose the maximum τ such that the average trigger delay is less than 15 min. Choosing a "best" trigger period to balance the trade-offs depends on the distribution of the trigger and other system requirements. We leave the parameter selection as future work.

For ATAP, the goodput (which is the same as the throughput in this case) dropped drastically by about 80%. We found that one of the performance bottlenecks is the re-randomization of action service ID on the trigger service. However, the action service ID is fixed for each rule and is not event-dependent, which indicates that the re-randomization does not need to be performed for each trigger and can be generated beforehand. Without re-randomization, our test results show that the goodput only decreases by 12.5%.

6 Discussion

Incentive of Adoption. Similar to anonymous communication, which cannot achieve strong anonymity, low bandwidth, low delay simultaneously as proved in [18], our solution does not aim at providing all of them. Instead, by presenting various modes and configurable parameters, our system allows users to best fit their requirements. For example, for delay-critical applications in trigger-action platform, the user can decide whether to activate our schemes for strong anonymity at the cost of performance.

From the perspective of trigger-action platforms and partner services, we acknowledge that adopting our schemes may increase their operating costs. With the increased interest in the pay-for-privacy model [19], business incentives may exist to offer privacy-enhanced trigger-action system for additional revenues.

Deployability. Though several changes are required to adopt our scheme, we will explain how the transition to our schemes can be made easier with the help of a proxy. In our schemes, the core functions of the partner services remain the same. Therefore, the adoption of our schemes could be done by having a proxy that sits in between the service and the platform and transforms the requests and responses. Since all partner services follow the same API specification, the proxy can be written once by the platform and distributed to all partner services. It should be published in an open-source manner and run by each partner service on its own, so that the proxy can be trusted.

For OTAP, as discussed in Sect. 4.1, the existing trigger-action platform can be reused, though all the data and parameters sent to the trigger-action platform must be encrypted by the user clients and partner services.

Compared to OTAP, ATAP requires modifying the platform, and a reimplementation is inevitable. However, since its function is simpler compared to the existing one, it is likely that it would not require excessive development effort.

As for the user-side, the key managements and encryption/decryption can be done by installing a local app or a browser plugin to ease the burden.

7 Related Work

Researchers have worked on the security and privacy issues in the trigger-action ecosystem. We first review the related work on the platforms, and review other parts of the ecosystem, including automation rules, and IFTTT.

Untrusted Trigger-Action Platform. Xu et al. [30] studied privacy leakage in smart homes to the trigger-action services and the mitigation. They proposed “Filter-and-Fuzz (F&F)” that filters out events unneeded by IFTTT and fuzzes the event data and their frequencies. This fuzzing component is somewhat similar to our OTAP scheme, as it randomizes the values of event data and sometimes sends fake triggers. However, their usage scenario is limited to Boolean or numerical event data fields and they only consider the cases where the trigger and action services are the same.

Fernandes et al. [20] studied the security of IFTTT’s OAuth-based authentication protocols, and they found that 75% of the tokens are over-privileged and can be exploited by the attackers to control users’ devices when the platform is compromised. They proposed a decentralized trigger-action platform framework (DTAP), which allows the use of fine-grained transfer tokens (XToken). The trigger-action platform stores only rule-specific tokens while the XToken is stored in the newly-introduced trusted-client, which is controlled by the user. Although DTAP does not provide data confidentiality and privacy, DTAP can be combined with our solution to enhance the system privacy and integrity.

Security and Privacy Concerns of Automation Rules. Bastys et al. [13,14] examined IFTTT applets that contain *filter code*, and found that such applets are susceptible to URL-based attacks, which can exfiltrate private information to a third-party when the applet is created by a malicious maker. A malicious applet could also lead to integrity and availability violations. Authors proposed FlowIT that can monitor and prevent malicious apps from being executed.

Surbatovich et al. [28] analyzed 19,323 IFTTT rules based on information-flow techniques. They defined a four point lattice that checks whether the information flows from a trusted source to an untrusted sink using static analysis. A series of work utilized dynamic analysis techniques such as model checking or symbolic execution to detect and fix insecure interactions between rules or to synthesize secure trigger-action programs [15,23–25,29,31].

IFTTT Ecosystem. Mi et al. [27] conducted an empirical study on IFTTT to understand its ecosystem and the performance of rule executions. They leveraged the self-implemented IFTTT server to profile the interaction among different entities. Their work provides a deeper understanding of the architecture and the execution path of a rule, which inspired our design.

8 Conclusion

Emerging trigger-action platforms empower users to conveniently combine various online services and physical devices for customized automation. However, their centralized design allows these platforms to collect personal information from multiple services, which raises privacy concerns. This work conducted the first empirical study on potential privacy exposures to trigger-action platforms, and presents two practical mitigations that enhance privacy without sacrificing the convenience promised by these platforms. In our empirical study of the 500 most popular IFTTT triggers, we found that the platform is capable of obtaining a variety of sensitive information, and 91% of the popular triggers are susceptible to privacy leakage. To mitigate the problem, we designed and implemented two privacy-preserving trigger-action platform systems, OTAP and ATAP. Trigger *obfuscation* and trigger *anonymization* techniques can hide trigger presence, so that the platform (1) sees real and fake triggers that are indistinguishable, or (2) cannot determine which user is related to a given trigger. We believe that our work provides an immediate remedy to enhance today's trigger-action platforms, and an interesting future direction is utilizing a clean-slate approach that ensures security and privacy by design.

Acknowledgments. This research was supported by the Ministry of Science and Technology of Taiwan under grants MOST 109-2636-E-002-021 and MOST 109-2636-E-005-002.

A Formal Security Analysis

Privacy Framework. We adopt the ideal/real-world paradigm to analyze the privacy of our schemes, which is standard in MPC literature [16,21,22]. Conceptually, there are two worlds, one is ideal and the other is real, both evaluating the functionality $\mathcal{F}_{tap}$, the trigger-action platform protocol. The "ideal-world" has a trusted party $\mathcal{T}$ carrying out all the computations. All other parties send their input to $\mathcal{T}$ and receive their prescribed output through a secure channel. In "real-world", no such party exists and all parties perform the computation themselves. To show that a real-world protocol is secure, we need to show that for every possible real-world adversary $\mathcal{A}$, there exists an ideal-world simulator $\mathcal{S}$ such that when controlling same parties as $\mathcal{A}$, the outputs of the protocols in ideal-world and real-world are computationally indistinguishable. This implies that every attack that can be done by $\mathcal{A}$ in real-world can be done by $\mathcal{S}$ in the ideal-world. Since the simulator learns nothing but the input/output of the corrupted parties, the real-world adversary $\mathcal{A}$ can only learn the same information.

Adopting this framework, the security of a privacy-preserving trigger-action system can be defined as follows.

Definition 1. *A trigger-action protocol Π that computes $\mathcal{F}_{tap}$ is secure if given any adversarial trigger-action platform $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that* $\mathrm{IDEAL}_{\mathcal{F}_{tap},\mathcal{S}} \approx \mathrm{REAL}_{\Pi,\mathcal{A}}$ *are computationally indistinguishable, where* $\mathrm{IDEAL}_{\mathcal{F}_{tap},\mathcal{S}}$ *is the joint output of simulator $\mathcal{S}$ in the ideal-world, while* $\mathrm{REAL}_{\Pi,\mathcal{A}}$ *is the output of adversary $\mathcal{A}$ in the real-world.*

Theorem 1. *Let Π be the protocol of OTAP and assume* $\mathsf{Enc}(\cdot)$ *is an IND-CPA encryption scheme. Given any adversarial trigger-action platform P, there exists* *a probabilistic simulator $\mathcal{S}$ which takes as inputs the rules sets $\boldsymbol{r}_1, \cdots, \boldsymbol{r}_M$ and the trigger periods $\boldsymbol{\tau}$ satisfies Definition 1.*

Proof. Let $I = \{(u, d_t) \in \boldsymbol{r}_i \mid i = 1, \ldots, M\}$ be the set of users and trigger parameters of rules, and $\tau_{(u,d_t)}$ be the associated trigger period for each $(u, d_t) \in I$. The simulator $\mathcal{S}$ will interact with the adversary $\mathcal{A}$ internally, and can be constructed as follows. For each $(u, d_t) \in I$, $\mathcal{S}$ passes (u, d_t, c) to $\mathcal{A}$ as input as if it was sent by the triggering service T at each corresponding trigger point (i.e., $\tau_{(u,d_t)}, 2 \cdot \tau_{(u,d_t)}, \ldots$) and outputs whatever $\mathcal{A}$ outputs, where c is randomly sampled from the ciphertext space of $\mathsf{Enc}(\cdot)$.

Now we show that the output of $\mathcal{S}$ is computationally indistinguishable from the output of $\mathcal{A}$ in real-world. In the real-world, the service T sends out $(u, d_t, \mathsf{Enc}(k, \delta))$ to $\mathcal{A}$ periodically for every $(u, d_t) \in I$, where δ is the event data. Based on the CPA secure assumption of Enc, we know that (u, d_t, c) and $(u, d_t, \mathsf{Enc}(k, \delta))$ are computationally indistinguishable. Thus, the outputs of $\mathcal{S} = \mathcal{A}(u, d_t, c)$ and $\mathcal{A}(u, d_t, \mathsf{Enc}(k, \delta))$ must also be computationally indistinguishable, which completes the proof.

Since the simulator is given only the rules sets $\boldsymbol{r}_i$ of every user i and the trigger periods, it is clear that the platform learns nothing about the trigger events, and thus preserved *trigger presence confidentiality* and *trigger data confidentiality*. However, *device set confidentiality* is broken, as the devices owned by a user can be inferred from the rules they are using.

Theorem 2. *Let Π be the protocol of ATAP, and $\mathcal{G}$ be the elliptic curve used in Π, whose base point being G and order being q. Assume that use of IND-CPA encryption schemes $\mathsf{Enc}(\cdot)$ and $\mathcal{E}(\cdot)$, then given any adversarial trigger-action platform $\mathcal{A}$, there exists a probabilistic simulator $\mathcal{S}$ satisfies Definition 1, while $\mathcal{S}$ takes as input a list $\boldsymbol{E}$ that contains the 3-tuples of events occurrence time, trigger service name, and action service name.*

Proof. The simulator $\mathcal{S}$ will interact with the adversary $\mathcal{A}$ internally, and can be constructed as follows. For each $(t, T, A) \in \boldsymbol{E}$, the simulator $\mathcal{S}$ passes $(\mathcal{E}_P(A), c, \gamma' G)$ to $\mathcal{A}$ as if it was sent by the trigger service T, where c is randomly picked up from the ciphertext space of Enc and γ is randomly chosen from $\{1, \ldots, q-1\}$. Then $\mathcal{S}$ simply outputs whatever $\mathcal{A}$ outputs.

Now we show that the output of $\mathcal{S}$ is computationally indistinguishable from the output of $\mathcal{A}$ in real-world. It is suffices to show that $(\mathcal{E}_P(A), c, \gamma' G)$ and $(\mathcal{E}_P(A), \mathsf{Enc}(k, \delta), \beta\gamma G)$ are computationally indistinguishable, following from the CPA security of underlying encryption schemes and the decisional Diffie-Hellman (DDH) assumption.

From the above proof we know that the platform learns nothing but the time of each event. However, since the platform will not know the name of the event and the associated user, the *trigger presence confidentiality* is still preserved. *Trigger data confidentiality* and *device set confidentiality* are also achieved, as no further information is provided to the platform.

B Triggering Delay Analysis

This subsection will show that the average delay for OTAP with the guaranteed delivery strategy is minimized when using a constant trigger period.

Consider a general case where the trigger period is not a constant but follows a distribution $\mathcal{I}$ with $E[\mathcal{I}] = \tau$. In the following, we focus on a particular trigger event and its corresponding service. We assume that the arrival of events is a Poisson process with a rate of λ. We model the trigger service of OTAP as a M/G/1 queueing system, where the customers (events) are served only at the trigger points and the service time is zero.

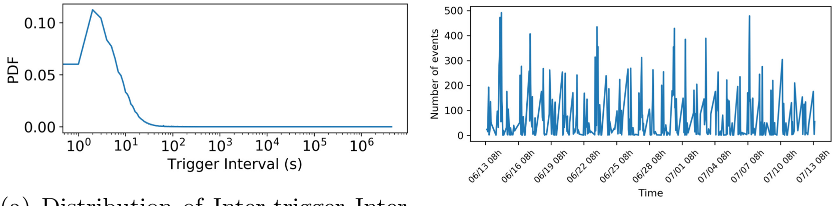
When an event arrives, it first needs to wait a short period of time before the queue starts serving new events, denoted as $\mathcal{R}$. Then the event needs to wait for all the events in the queue to be served before it is sent out. The duration is the sum of $\mathcal{N}_Q$ trigger periods, where $\mathcal{N}_Q$ denotes the size of the queue when the event arrives. As a result, we can derive the average delay of an event: $E[\mathcal{D}] = E[\mathcal{R}] + E[\mathcal{N}_Q] \cdot E[\mathcal{I}]$. By Little's Law [26], we have $E[\mathcal{N}_Q] = \lambda E[\mathcal{D}]$, and from queueing theory, we know that $E[\mathcal{R}] = \frac{\lambda}{2} E[\mathcal{I}^2]$, which implies

$$E[\mathcal{D}] = \frac{E[\mathcal{R}]}{1 - \lambda E[\mathcal{I}]} = \frac{\lambda E[\mathcal{I}^2]}{2(1 - \lambda\tau)} = \frac{\lambda(\mathrm{Var}[\mathcal{I}] + (E[\mathcal{I}])^2)}{2(1 - \lambda\tau)} = \frac{\lambda(\mathrm{Var}[\mathcal{I}] + \tau^2)}{2(1 - \lambda\tau)}$$

The result shows that the average delay is minimized when the trigger period is constant, since the minimum of $\mathrm{Var}[\mathcal{I}]$ $(= 0)$ happens when $\mathcal{I}$ is constant.

C Trigger Distribution

Figure 8a shows the distribution of the intervals between consecutive triggers and Fig. 8b shows the number of trigger events in each hour from the first 720 h of the CASAS hh104 dataset [17]. The occurrence of events is not uniformly distributed. Since the sensor only monitors motions within a specific area, a series of bursty triggers occurs when the user is present in that area, and no trigger occurs when the user is elsewhere.



(a) Distribution of Inter-trigger Intervals

(b) Number of Triggers in Each Hour

Fig. 8. Trigger distribution in the CASAS hh104 dataset

D Supporting Filter Code

As mentioned in Sect. 3.1, some trigger-action platforms allow users to write code to run during rule execution. Our schemes are capable of supporting a filter code by letting the action services store and run such codes. However, it might not be ideal to put this workload on action services instead of the trigger-action platform. Therefore, we propose two potential solutions to support this feature and leave them as future work.

The first direction is to replace the current encryption scheme with Homomorphic Encryption (HE), which allows computation over ciphertext without knowing the underlying plaintext. However, a challenge is achieving acceptable performance as HE schemes are computationally expensive.

Another possible solution is to build the trigger-action platform on trusted hardware. Intel Software Guard Extension (SGX), for example, provides an isolated execution environment called an enclave whose contents are protected, and only processes running inside the enclave are allowed access. Since the protection is at the hardware level, a malicious OS cannot read those data. SGX also provides another mechanism called remote attestation, which allows the client to attest that the code running on the remote machine is indeed the expected one. Thus, an SGX-based solution would support the trigger-action platform to handle sensitive data inside an enclave and partner services to attest that the platform is indeed running the privacy-preserving version at each interaction. However, the resources inside an enclave are limited and hence, a challenge becomes designing an algorithm that can use memory effectively.

References

1. Microsoft Flow. https://flow.microsoft.com
2. Target Expects $148 Million Loss from Data Breach (2014). https://time.com/3086359/target-data-breach-loss/
3. Yahoo Says Hackers Stole Data on 500 Million Users in 2014 (2016). https://www.nytimes.com/2016/09/23/technology/yahoo-hackers.html
4. Yahoo Triples Estimate of Breached Accounts to 3 Billion (2017). https://www.wsj.com/articles/yahoo-triples-estimate-of-breached-accounts-to-3-billion-1507062804
5. ApacheBench (2019). https://httpd.apache.org/docs/2.4/programs/ab.html
6. Do more with Oticon - IFTTT (2019). https://ifttt.com/oticon
7. IFTTT (2019). https://ifttt.com
8. IFTTT Platform (2019). https://platform.ifttt.com/lp/learn_more
9. The Equifax Data Breach (2019). https://www.ftc.gov/equifax-data-breach
10. Turn off WiFi on your Android when you leave home to save power (2019). https://ifttt.com/applets/302237p
11. Zapier (2019). https://zapier.com
12. (2020). https://github.com/csienslab/tap-privacy
13. Bastys, I., Balliu, M., Sabelfeld, A.: If this then what?: controlling flows in IoT apps. In: ACM CCS (2018)

14. Bastys, I., Piessens, F., Sabelfeld, A.: Tracking information flow via delayed output. In: Gruschka, N. (ed.) NordSec 2018. LNCS, vol. 11252, pp. 19–37. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03638-6_2
15. Bu, L., et al.: Systematically ensuring the confidence of real-time home automation IoT systems. ACM Trans. Cyber-Phys. Syst. **2**(3), 1–23 (2018)
16. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. CRYPTOLOGY **13**(1), 143–202 (2000)
17. Cook, D.J., Crandall, A.S., Thomas, B.L., Krishnan, N.C.: CASAS: a smart home in a box. Computer **46**(7), 62–69 (2012)
18. Das, D., Meiser, S., Mohammadi, E., Kate, A.: Anonymity trilemma: strong anonymity, low bandwidth overhead, low latency-choose two. In: IEEE Symposium on Security and Privacy (S&P) (2018)
19. Elvy, S.A.: Paying for privacy and the personal data economy. Columbia Law Rev. **117**, 1369 (2017)
20. Fernandes, E., Rahmati, A., Jung, J., Prakash, A., Rahmati, A.: Decentralized action integrity for trigger-action IoT platforms. In: NDSS (2018)
21. Goldreich, O.: Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press, Cambridge (2009)
22. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: ACM Symposium on Theory of Computing (1987)
23. Hsu, K.H., Chiang, Y.H., Hsiao, H.C.: SafeChain: securing trigger-action programming from attack chains. IEEE Trans. Inf. Forensics Secur. **14**(10), 2607–2622 (2019)
24. Liang, C.J.M., et al.: Systematically debugging IoT control system correctness for building automation. In: ACM BuildSys (2016)
25. Liang, C.J.M., et al.: Sift: building an internet of safe things. In: IEEE/ACM IPSN (2015)
26. Little, J.D.: A proof for the queuing formula: L = λW. Oper. Res. **9**(3), 383–387 (1961)
27. Mi, X., Qian, F., Zhang, Y., Wang, X.: An empirical characterization of IFTTT: ecosystem, usage, and performance. In: ACM IMC (2017)
28. Surbatovich, M., Aljuraidan, J., Bauer, L., Das, A., Jia, L.: Some recipes can do more than spoil your appetite: analyzing the security and privacy risks of IFTTT recipes. In: International Conference on World Wide Web (2017)
29. Wang, Q., Datta, P., Yang, W., Liu, S., Bates, A., Gunter, C.A.: Charting the attack surface of trigger-action IoT platforms. In: ACM CCS (2019)
30. Xu, R., Zeng, Q., Zhu, L., Chi, H., Du, X., Guizani, M.: Privacy leakage in smart homes and its mitigation: IFTTT as a case study. IEEE Access **7**, 63457–63471 (2019)
31. Zhang, L., He, W., Martinez, J., Brackenbury, N., Lu, S., Ur, B.: AutoTap: synthesizing and repairing trigger-action programs using LTL properties. In: IEEE/ACM ICSE (2019)