

ACACIA: A Practical, Privacy-Preserving Trigger-Action Platform using Multi-Party Computation

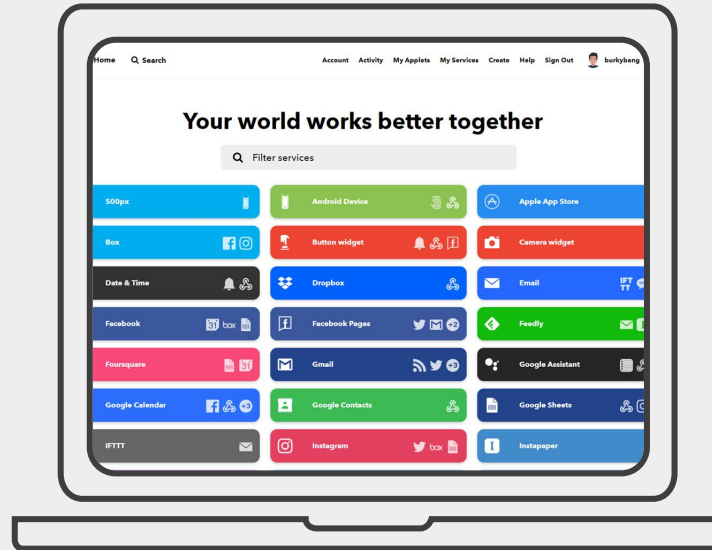
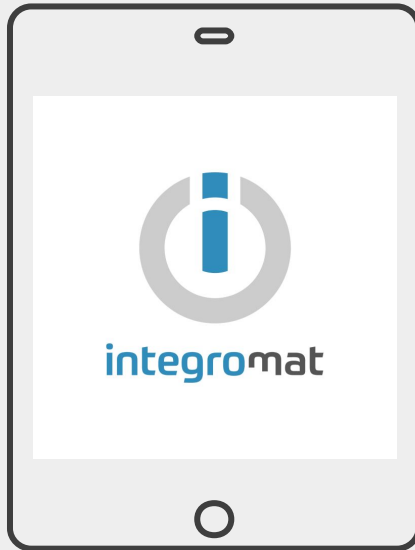
By Veronika Kitsul, mentor: Kevin Liao at MIT



Trigger-Action Platforms (TAPs)



Market Analysis



Today's TAPs

Plans	Charge	Zaps	Tasks / month	Update Time	Premium Applications
Free	0 \$ Free	5 Single-step	100	15 mins	-
Starter	19.99 \$ annual 24.99 \$ mon by mon	20 Multi-step	750	15 mins	3
Professional	49 \$ annual 61.25 \$ mon by mon	unlimited Multi-step	2000	2 mins	unlimited
Team	299 \$ annual 373.75 \$ mon by mon	unlimited Multi-step	50,000	1 min	unlimited
Company	599 \$ annual 784.75 \$ mon by mon	unlimited Multi-step	100,000	1 min	unlimited

[Zapier templates](#)

Zapier pricing plans [1].



Gmail + Facebook Pages

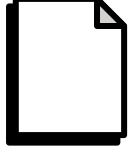
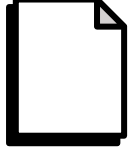
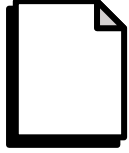
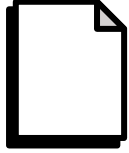
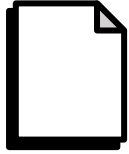
Post new emails received in Gmail [Business Gmail Accounts Only] to Facebook



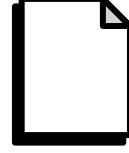
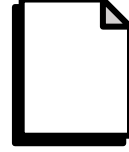
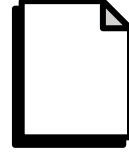
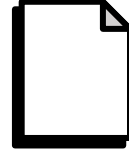
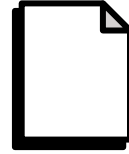
When this happens
Step 1: New Email



Then do this
Step 2: Create Page Post



- the services (both trigger and action, for example Gmail and Facebook),
- the recipe semantics ("If new email – post it to a Facebook page."),
 - inputs to the applet (email), and
 - outputs to the action service (the Facebook post),
- improve performance.



Multi-Party Computation

Assumes direct secure channels between each pair and denotes encryption and decryption of a message m under key κ as $\text{Enc}_{\kappa}(m)$ and $\text{Dec}_{\kappa}(m)$ with the goal to learn the correct output of a mutual function without revealing private inputs.



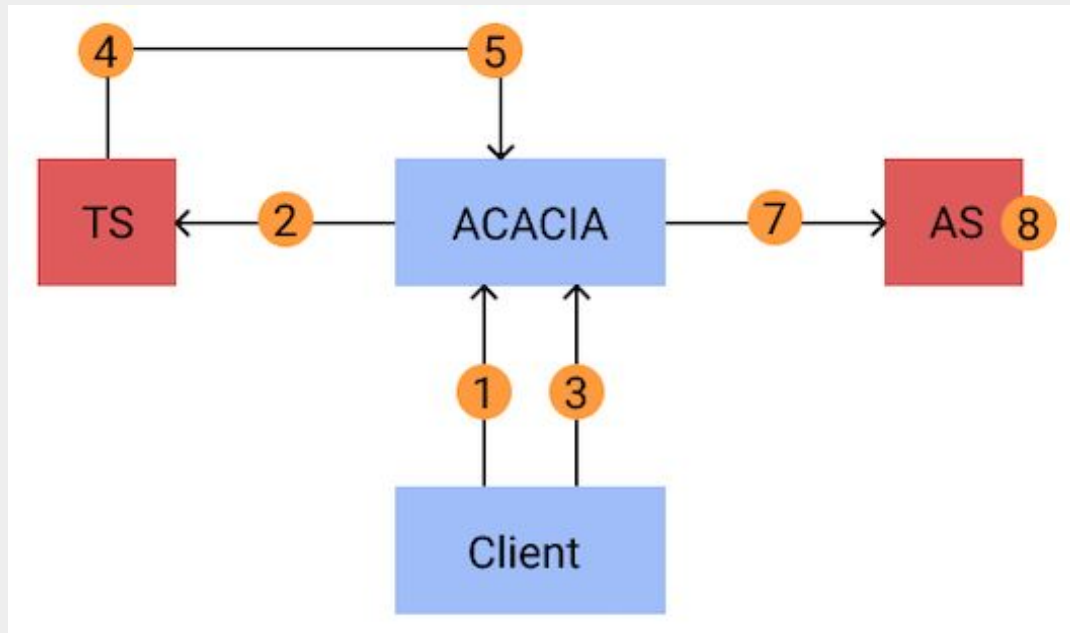
Garbled Circuits

Yao's Garbled Circuits operate on the idea that there is a function $F(x,y)$, party $P1$ holds $x \in X$, and party $P2$ holds $y \in Y$.

Real-ideal Paradigm

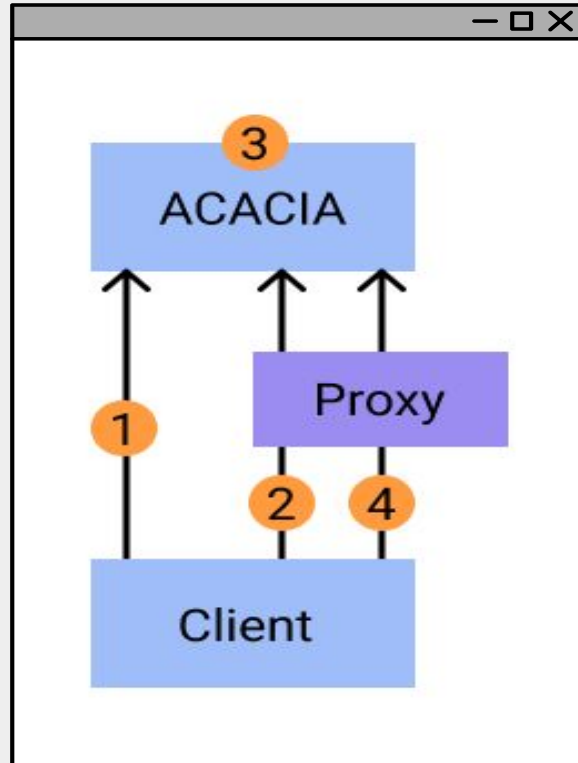
$\text{Real}_{\pi}(\kappa, C, x_1, \dots, x_n)$
 $\text{Ideal}_{\phi}, \text{Sim}(\kappa, C, x_1, \dots, x_n)$
are indistinguishable in κ .

Basic Protocol



Basic protocol recipe installation scheme. (1) - recipe installation; (2) - send encrypted recipe information; (3) - generate and send garbled circuits; (4) - get trigger data xt and execute $P(xt, ct)$; (5) - garbled inputs for $f(xt, ct)$; (6) - execute $f(xt, ca)$; (7) - send y ; (8) - decrypt y to get action data.

Anonymity Extension



Anonymous recipe instalation protocol scheme. (1) - Get anonymous tokens; (2) - send recipe installation data with anonymous token; (3) - check token validity, install the recipe if valid; (4) - upload garbled circuits with authorization token. The rest of the steps follow the scheme in Figure 1.

Implementation

Rust:

- Fast
- Extensive support
- Efficient
- High safety

C++ MPC library:

<https://github.com/emp-toolkit/emp-sh2pc>

Java mobile client.

```
fn encrypt_msg(  
    msg: &[u8],  
    aad: &[u8],  
    server_pk: &<Kem as KemTrait>::PublicKey,) -> (<Kem as KemTrait>::EncryptedKey, <Kem as KemTrait>::Tag)  
    let mut csprng = StdRng::from_entropy();  
  
    // encapsulate a key and use the resulting shared secret to encrypt the message  
    // encrypt with AEAD context  
    let (encapsulated_key, mut sender_ctx) = hpke::setup_sender::(<Kem as KemTrait>::Base, &mut csprng).expect("failed to setup sender");  
  
    // seal in place will encrypt the plaintext in place if successful  
    let mut msg_copy = msg.to_vec();  
    let tag = sender_ctx.seal_in_place_detached(&mut msg_copy, aad);  
  
    let ciphertext = msg_copy;  
    println!("ciphertext: {:?}", ciphertext);  
    // return  
    (encapsulated_key, ciphertext, tag)  
}
```

Evaluation Metrics

What are the computation and communication costs for each of the parties in our system?

How do the added privacy and anonymity guarantees affect the latency and throughput of our system?

Throughput

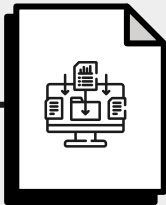
Latency



Future Work Directions



Running the
system on
real-world
workloads and
comparing with
current TAPs



Tokens and
proxy
improvement



Anonymity
without a proxy



Better
performance,
general mpc
betterment

Conclusion

```
use std::io::{Read, Write};
use std::net::TcpStream;
use example::format::{Message, MessageType};
use bincode;
use rand::prelude::*;
use hpke::{
    AeadTag, ChaCha20Poly1305,
    Kdf: HkdfSha384,
    Kem: X25519HkdfSha256,
    Deserializable, Kem as KemTrait, OpMode, OpModes, Serializable,
};

type Kem = X25519HkdfSha256;
type Aead = ChaCha20Poly1305;
type Kdf = HkdfSha384;

const INFO_STR: &[u8] = b"example session";

// initialize the server with a key pair
// can write it in a file serialized and then use it as client and deserialize it
fn server_init() -> (<Kem as KemTrait>::PrivateKey, <Kem as KemTrait>::PublicKey) {
    let mut csprng = StdRng::from_entropy();
    Kem::gen_keypair(&mut csprng)
}

fn encrypt_msg(
    msg: &[u8],
    aad: &[u8],
    server_pk: <Kem as KemTrait>::PublicKey,
) -> (<Kem as KemTrait>::EncappedKey, Vec<u8>, AeadTag) {
    let mut csprng = StdRng::from_entropy();

    // encapsulate a key and use the resulting shared secret to encrypt the message
    // encrypt with AEAD context
    let (encapsulated_key, mut sender_ctx) = hpke::setup_sender::<Aead, Kdf, Kem>
        (&OpModes::Base, &server_pk, &mut csprng).expect("invalid server pubkey!");

    // seal in place will encrypt the plaintext in place if success
    let mut msg_copy = msg.to_vec();
    let tag = sender_ctx.seal_in_place_detached(&mut msg_copy, aad).expect("encryption failed!");

    let ciphertext = msg_copy;
    println!("ciphertext: {:?}", ciphertext);
    // return
    (encapsulated_key, ciphertext, tag)
}
```

```
example > src > bin > @ server.rs > ...
1 use std::net::{TcpListener, TcpStream, Shutdown};
2 use std::thread;
3 use std::io::{Read, Write};
4 use hpke::{
5     AeadTag, ChaCha20Poly1305,
6     Kdf: HkdfSha384,
7     Kem: X25519HkdfSha256,
8     Deserializable, Kem as KemTrait, OpMode, OpModes, Serializable,
9 };
10
11 fn decrypt_msg(
12     ciphertext: Vec<u8>,
13     encapsulated_key: <Kem as KemTrait>::EncappedKey,
14     aad: &[u8],
15     tag: AeadTag,
16 ) -> (<Kem as KemTrait>::PrivateKey, <Kem as KemTrait>::PublicKey) {
17     // ...
18     let (server_sk, server_pk) = (<Kem as KemTrait>::PrivateKey::from_bytes(server_sk_bytes).
19         .expect("could not deserialize server privkey"),
20         <Kem as KemTrait>::PublicKey::from_bytes(server_pk_bytes).expect("could not deserialize AEAD tag"));
21     let decapped_key = <Kem as KemTrait>::DecappedKey::from_bytes(encapsulated_key_bytes).
22         .expect("could not deserialize the encapsulated pubkey");
23
24     // calculate and derive the shared secret to create AEAD context
25     let receiver_ctx = hpke::setup_receiver::<Aead, Kdf, Kem>
26         (&OpModes::Base, &server_pk, &decapped_key, &INFO_STR).expect("failed to set up receiver!");
27
28     let mut plaintext = ciphertext.to_vec();
29     receiver_ctx.open_in_place_detached(&mut plaintext, aad, &tag).expect("invalid ciphertext");
30
31     let mut plaintext = plaintext;
32     let aad = b"First encrypted message";
33
34     // Marshall the message into bincode
35     let serialized: Vec<u8> = bincode::serialize(&plaintext).unwrap();
36     // looks good
37     println!("serialized: {:?}", serialized);
```



References

[1] Mohammed Abdou, Abdelrahman M.Ezz, and Ibrahim Farag. 2021. Digital Automation Platforms Comparative Study. In 4th International Conference on Information and Computer Technologies, ICICT 2021, Kahului, HI, USA, March 11-14, 2021. IEEE, 279–286. [https://doi.org/ 10.1109/ICICT52872.2021.00052](https://doi.org/10.1109/ICICT52872.2021.00052)

[2] Yunang Chen, Amrita Roy Chowdhury, Ruizhe Wang, Andrei Sabelfeld, Rahul Chatterjee, and Earlence Fernandes. 2021. Data Privacy in Trigger-Action Systems. In 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021. IEEE, 501–518. <https://doi.org/10.1109/SP40001.2021.00108>

[3] SandySchoettler,AndrewThompson,RakshithGopalakrishna,and Trinabh Gupta. 2020. Walnut: A low-trust trigger-action platform. CoRR abs/2009.12447 (2020). arXiv:2009.12447 [https://arxiv.org/abs/ 2009.12447](https://arxiv.org/abs/2009.12447)

References

[4] [n.d.]. IFTTT. <https://ifttt.com/>

[5] David Evans, Vladimir Kolesnikov, and Mike Rosulek. 2018. A Pragmatic Introduction to Secure Multi-Party Computation. Found. Trends Priv. Secur. 2, 2-3 (2018), 70–246. <https://doi.org/10.1561/33000000019>

[6] Earlence Fernandes, Amir Rahmati, Jaeyeon Jung, and Atul Prakash. 2017. Decoupled-IFTTT: Constraining Privilege in Trigger-Action Platforms for the Internet of Things. CoRR abs/1707.00405 (2017). arXiv:1707.00405 <http://arxiv.org/abs/1707.00405>

[7] Rixin Xu, Qiang Zeng, Liehuang Zhu, Haotian Chi, Xiaojiang Du, and Mohsen Guizani. 2019. Privacy Leakage in Smart Homes and Its Mitigation: IFTTT as a Case Study. IEEE Access 7 (2019), 63457–63471. <https://doi.org/10.1109/ACCESS.2019.2911202>

Find more information in the report paper.