TagAlong: Free, Wide-Area Data-Muling and Services

Alex Bellon
abellon@ucsd.edu
UC San Diego

Alex Yen
alyen@ucsd.edu
UC San Diego

Pat Pannuto
ppannuto@ucsd.edu
UC San Diego

ABSTRACT

We demonstrate how to leverage Apple's Find My protocol, most well known as the underlying protocol of the AirTag, for arbitrary data-muling and location services. This provides a new "infrastructure-free" deployment, where areas with frequent human activity can take advantage of this zero-cost backhaul network. While there are severe limitations (e.g. no acknowledgement channel back to the sending device), Find My-based networking could still be a reliable backhaul with sufficient transmission redundancy and knowledge of deployment context. Towards that end, we develop TagAlong, a protocol for scalable, efficient data transmission on the Find My network. We implement a proof-of-concept and demonstrate throughput up to 12.5 bytes/sec and up to a 97% data reception rate.

CCS CONCEPTS

• **Networks** → *Mobile and wireless security*; **Mobile ad hoc networks**.

KEYWORDS

AirTag, Apple, Bluetooth, tracking tags, data transport

ACM Reference Format:
Alex Bellon, Alex Yen, and Pat Pannuto. 2023. TagAlong: Free, Wide-Area Data-Muling and Services. In *The 24th International Workshop on Mobile Computing Systems and Applications (HotMobile '23), February 22–23, 2023, Newport Beach, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3572864.3580342

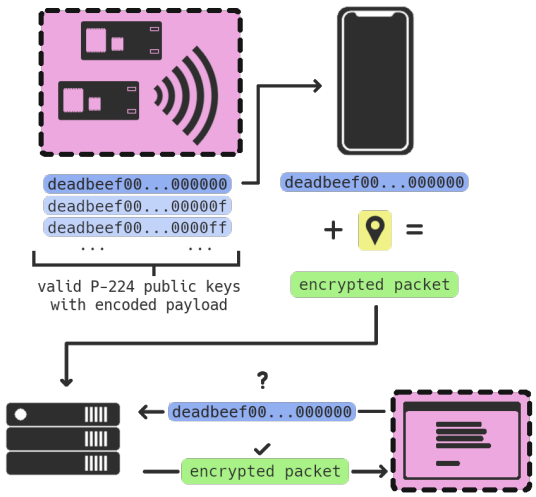

Figure 1: TagAlong Overview Deployed sensors encode arbitrary data in a series of Find My-compliant advertisements. Nearby Apple devices hear these, attach their current GPS location, and upload the advertisements to Apple's servers. Our TagAlong service queries Apple's servers and decodes the data. The hardware under TagAlong's control are highlighted in pink, while the rest of the hardware is controlled by Apple.

1 INTRODUCTION

For widespread wireless sensor networks, data backhaul is consistently one of the most challenging and costly deployment hurdles. State of the art often boils down to manually deploying gateway infrastructure for each network—indeed, the Internet of Things (Still) Has a Gateway Problem [13, 15].

Smartphones have consistently held appeal as a potential solution: they are ubiquitous, are sufficiently resource-capable to service many leaf devices, and are managed by invested users who keep them powered and online. Early work using dedicated phone apps as data mules demonstrated feasibility, but the scope of such muling infrastructure is inherently limited to individuals who elect to download the app [10]. If such an app were built-in to the phone, however, suddenly the coverage is all smartphone users. For a brief window of time, this happened with Google's Physical Web service running on all Android phones, and the Freeloader project showed how to exfiltrate ~1 bit/sec via Eddystone [1]. With the launch of AirTags and the widespread deployment of Apple's Find My service, the window is open again and wider: We show how to use the Find My protocol as a general-purpose data and services channel for Bluetooth-enabled sensor nodes, providing both location services and a data rate of up to 12.5 bytes/sec.

The intended functionality of AirTags and the Find My protocol is to track items that can be lost (e.g. keys, wallet). AirTags themselves are Bluetooth Low Energy (BLE) devices that broadcast BLE advertisement packets, which are in turn picked up by nearby Apple devices that have the "Find My" application installed.[1] These nearby Apple devices add their GPS location and forward along the original advertisement to Apple's servers for later access. To protect user privacy, Find My advertisements encrypt their identity in such a way that the muling device and Apple's servers cannot learn anything about the advertising device. This same privacy-protection is what enables arbitrary data-transmission. Mules do not know they are ferrying data. It simply appears that there are a large number of Find My-enabled devices in the area.

We are not the first to exploit the Find My protocol for purposes and devices beyond its originally intended use. Much of this work

[1] Find My is installed and active by default on all contemporary Apple devices.

HotMobile '23, February 22–23, 2023, Newport Beach, CA, USA

ACM ISBN 979-8-4007-0017-0/23/02.
https://doi.org/10.1145/3572864.3580342

stands on the shoulders of the OpenHaystack project [8] and the team's prior reverse-engineering efforts on Find My [2]. Similarly, Positive Security's SendMy established the feasibility of data exfiltration via Find My [3]. With TagAlong, we investigate what it would look like to build an actual deployment atop Find My. What throughput, what reliability can this one-way, ALOHA channel achieve? When should devices send messages to maximize reception rate? How do we scale to thousands of devices sending arbitrary volumes of data? How does TagAlong compare to more traditional networking? This paper explores these questions and aims to establish the viability of Find My-based networking.

2 BACKGROUND AND RELATED WORK

We establish the background and context needed to understand how this work builds off of previous work [3]. We then discuss prior efforts towards commodity wide-area connectivity.

2.1 Background

AirTags (and other Find My-enabled devices) are meant to be found when they are "lost". When not near an 'owning' device, they beacon BLE advertisements, which are picked up by Apple devices that relay these "lost" announcements to Apple's Find My network. Once paired with an Apple device, AirTags broadcast P-224 public keys derived from a key pair established during the pairing process. When any Apple device picks up a Find My-compliant advertisement, it encrypts its own GPS location with a key based on the advertised public key, and sends the encrypted data—a "location report"—to Apple's servers. The data is then stored in a hash map which maps the hashed public key to the encrypted location report. 'Lost device' owners then query for data from their AirTag based on the rolling public keys that could have been generated.

Heinrich et al. questioned the security and authentication of this system and found that they can use arbitrary BLE devices to pose as AirTags; this provides the first insight that the Find My network is not secure [9]. Positive Security then realized that if any BLE device can send public keys—under ECC encryption—to the Find My network servers, then there is a possibility to send and *receive* data through the Find My network. They show that one can send and retrieve arbitrary messages through BLE advertisements. We build off of their initial implementation to send data at a much higher rate through a new encoding and decoding scheme.

2.2 Related Work

It is a monumental effort to provide wide-area coverage. For instance, cellular providers deploy (and *maintain*) cell towers to ensure reliable coverage throughout most inhabited parts of the world. In lieu of costly fixed infrastructure, several prior systems perform data muling for sensors nodes with smartphones [4, 10, 12]—when users install their app.

Success at scale requires buy-in from the general population...or the device manufacturers who control the OS and firmware on phones for the general population. Indeed, when Android elected to silently fetch the UI of Physical Web devices, Adkins et al. showed how to use the (now decommissioned) Nearby Notifications to transport arbitrary data via Android phones [1]. Through unintended use of Nearby Notification's BLE packets, they showed that they were able to transmit data at an average data rate of 0.1-2.6 bps.

2.3 "Send My"

Positive Security implemented an expanded AirTag functionality that allowed arbitrary data transmission through their "Send My" project [3]. Send My allows ESP32 microcontrollers to act as AirTag-like devices, using surrounding Apple devices to transmit data bit-by-bit in Find My-compliant BLE advertisement packets. The 'packet' structure embeds a device ID, data index, and one new bit of information in each advertisement. The data can then be downloaded through Positive Security's "DataFetcher" application, which is built on top of OpenHaystack [7, 8]. While effective as a proof-of-concept for the notion of data exfiltration via Find My, Send My was only ever intended as exactly that: proof it can be done. With TagAlong, we ask how to do it well.

TagAlong is designed to scale to realistic sensor deployments with an arbitrary number of devices deployed for an unbounded duration without limits on message quantity or frequency from any one device. It achieves greater bandwidth of data muling using Apple devices and considers techniques to realize an approximation of reliable transmission on a unidirectional, broadcast-only channel.

2.4 Advertisement-Based Networking

Recent work has shown that with modest repetition, advertisements can be a reliable medium [14]. A network of over one hundred power meters deployed one home found that even in this dense context, robust data recovery with advertisement-based networking is feasible [5]. We incorporate these lessons in TagAlong and include modest repetition to achieve robust transmission. We explore the repetition factor with unplanned, not-100%-duty-cycle Apple devices as scanners further in our evaluation.

2.5 Example Walkthrough

We present a walkthrough of the TagAlong protocol using the example from Figure 2 to further clarify the process.

3 THE TAGALONG PROTOCOL DESIGN

The goal of TagAlong is to use the Find My network to provide services 'for free' to wireless sensor nodes. In particular, we show how to provide location and data muling services. As shown in Figure 1, the TagAlong system encodes arbitrary data in the P-224 public keys contained in standard Find My BLE advertisements packets. Nearby Apple devices, believing these advertisements to be real Find My messages, will follow the usual process for handling Find My packets. They will encrypt their GPS location using the information in the Find My advertisement to construct a location report and will upload this to Apple's servers. Location reports are stored in a hash table on Apple's servers, keyed with the (hash of the) P-224 public key from the Find My BLE advertisement packet. The server provides an API to query for specific P-224 public keys to retrieve their associated location reports; this API—in conjunction with our custom TagAlong packet format—can be used as a sort of oracle to progressively decode the message originally encoded in the BLE advertisement.

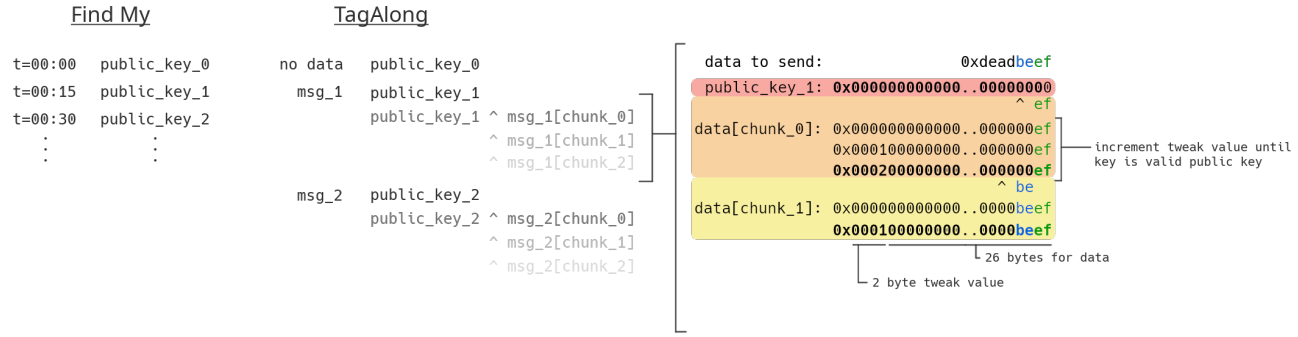

Figure 2: The TagAlong Protocol. TagAlong uses standard Find My primitives to uniquely identify each sensor device. Instead of varying the ephemeral key by time, TagAlong uses new ephemeral keys to signal a new message. Data is the chunked into n bits; $n = 8$ in this example. Each chunk is encoded by XOR-ing the prior key with the new data, and then a tweak value is incremented until the public key becomes a valid P-224 key. Progressive data chunks 'slide' along the key to create a unique path through the keyspace for this payload. In this figure, each colored rectangle represents one iteration of calculating the public key to advertise, and the bolded key is what is transmitted.

An overview of the TagAlong encoding protocol is shown in Figure 2, which also compares TagAlong to normal AirTag functionality. Succinctly, sensors use the out-of-the-box Find My protocol (albeit with modified timing) to uniquely identify themselves and to indicate a new data transmission by rolling to the next ephemeral Find My device key. TagAlong then encodes and later decodes data by using knowledge of the previously sent public key. We continuously XOR the 28 byte public key between the current and the previous key with new data. The result of the XOR is fragmented payload data that is used to create the full message. This technique reduces the need to query and search for random public keys that *might* have been received by the Find My network servers.

3.1 Link Layer Limitations

FindMy uses non-connectable BLE advertisements. As a consequence, TagAlong is necessarily a uni-directional communication medium—there is no acknowledgement channel. This is consistent with the baseline functionality of AirTags and other devices on the Find My network, which also have no acknowledgement of received packets. While additional infrastructure could provide downlink services, we are interested in understanding what can be done using the Find My network as it is deployed today without additional deployment-specific infrastructure.

3.2 How to Encode Data?

Uncontrolled mules move in unknown ways. Thus, wireless sensor nodes need to send data quickly. Prior work by Positive Security encoded a changeable device ID in each advertisement [3]; this has very high overhead. Instead, we rely on the original Find My protocol which uses an Apple device's private key to generate ephemeral public keys; each public key serves as a unique ID for a particular BLE device. Using this ephemeral key as a root, we then XOR chunks of data using the process illustrated in Figure 2. This new value is likely not a valid P-224 public key and would be discarded by Apple data mules. Thus, a "tweak" field is added, which increments until a valid public key is found. This data chunk is advertised, and then the process repeats until the whole message is sent.

3.3 Identifying Devices & Unique Messages

If every recoverable advertisement must be a unique P-224 public key, how can the sender be identified? Fixing bytes of the key as 'device ID' bytes (a la Send My) limits both the space of possible valid public keys and caps the number of unique devices deployable *globally*. Instead of trying to embed IDs, TagAlong recognizes that this problem is already solved by the original Find My protocol. Recall, its whole purpose is to find lost devices.

The rolling public keys normally generated by AirTags can be used as a unique device identifier. The original Find My design updates the key once a day and least significant bytes every fifteen minutes. TagAlong builds off of Find My's design and uses the daily rolling public keys to indicate device liveness and uses the least-significant-bytes incrementation to indicate the start of a new message. The data recovery system begins with a Find My query as originally designed to assess whether (1) the device has been seen at all in the last day and (2) how many messages a device has sent in a given day. For each message, the data recovery system will then query for subsequent keys that make up the data for that message.

3.4 Ferrying the Data

For each chunk of data, sensors broadcast out a Find My-compliant BLE advertisement with the encoded data embedded inside the P-224 public key. Any Apple device that is nearby (and has opted-in to the Find My network) is continuously scanning for Find My advertisements. Since the TagAlong BLE advertisements conform to the format of a normal Find My advertisement packet, nearby Apple devices will process them as they would any other Find My packet, unaware of the data encoded in TagAlong's packets. Following the standard Find My process, the Apple device will create an encrypted location report—described in detail by Heinrich et al. [9]—and then upload it to Apple's Find My servers, where it is stored in a table keyed on the P-224 public key. Every Apple device that picks up the initial Find My BLE advertisement will create and upload a location report. So, for a given P-224 public key broadcasted by a Find My device, there may be multiple location reports associated with it.

3.5 How to Discover and Recover Data?

Apple provides an API that allows authenticated Apple accounts to query the Find My servers for specific P-224 public keys to get the associated location reports. In standard Find My, the P-224 public keys included in the AirTag's Find My packets are calculated deterministically. When an owner wants to find their AirTag, they query this API with a time range to look for location reports as well as a list of the hashes of P-224 public keys that the client wants location reports for. Any location reports that exist for a given public key are then returned. It is up to the client to decrypt and extract the GPS location to locate their device.

For TagAlong, the key part of this system is that the API only returns location reports if the queried-for public key exists on the server. TagAlong uses this property to craft a pseudo-oracle out of the API: to determine the next chunk of the encoded message, simply query the API for all possible values for that chunk (i.e. for the ith chunk of the message, calculate all 2^n possible public keys for the chunk size n). Whichever public key has the most location reports is the public key encoding the next chunk of data. Since the API takes in a list of possible public keys as input, there is only one API query per chunk containing all possible public keys. In our testing we were never rate-limited with our queries, even when sending hundreds of queries per minute. If Apple were to institute rate-limiting on the number of queries, one could simply increase the chunk size in order to get more data per query and space out the queries temporally to account for the increased time to calculate all 2^n possible keys for a larger n.

A normal AirTag broadcasts a new public key in its BLE advertisements every 15 minutes when in "lost" mode [9], calculated deterministically from the root keypair it shares with the owning device. TagAlong instead uses these public keys (`public_key_0`, `public_key_1`, etc. in Figure 2) to denote the start of new messages from a specific edge device ("modem"). To begin, a TagAlong device will broadcast `public_key_0`. Since the public keys are deterministically generated, the desktop client using TagAlong will be able to calculate `public_key_1` and query the Find My API for it. The edge device has not yet broadcast any messages, so the Find My API will not return any location reports for `public_key_1`.

When the edge device is ready to send a new message, it broadcasts `public_key_1`. It then takes the payload to be encoded and adds a terminating `0x00` character, converts it to a bitstring, divides that bitstring into chunks of size n, and pads the data to the correct length with zeroes; in this case, $n = 8$ for simplicity. Starting with the end-most chunk—`chunk_0`—and working left chunk by chunk, the current chunk, `chunk_i`, is XORed with the previous chunk's encoded public key. For the `chunk_0`, this is `public_key_1`, which is XORed with `0xef`. The "public key" that we have filled with encoded data might no longer be a valid P-224 public key, so we increment a counter in the front of the "public key" until it is a valid P-224 public key. Once valid, it is broadcasted in a properly formatted Find My BLE advertisement packet and the process is repeated for the entire payload.

Meanwhile, the desktop application's query for `public_key_1` will succeed once the BLE advertisement denoting the start of a new message gets to Apple's servers. The desktop application will begin decoding the message by mirroring the process the edge device follows to encode and send the message. It will search for the end-most chunk by calculating all $2^n = 2^8 = 256$ possible values for the first data-bearing public key the edge device could have sent. Note that this search must both XOR possible data and increment the tweak value appropriately: `0x00...00`, `0x00...01`, up to `0x00...ff`. One of these calculated keys will be the correct public key, `0x0002...ef`. While calculating all the possible public keys, it will also calculate the corresponding chunk value for each possible public key so it does not have to extract that data from the location report. The application will then query the Find My API for the list of all 256 calculated possible public keys. The response from the API should be a large number of location reports for the correct public key, `0x0002...ef`, and either zero or a small number of reports for any other public keys (due to random bit errors/actual AirTag public keys that happen to collide). The public key with the most associated location reports represents the encoded chunk of data, and the work begins on the next chunk. This process ends once a `0x00` byte is decoded; the bitstring is completely recovered and can be converted back to the actual payload.

3.6 Corner Cases

TagAlong does not encode indices for the chunks of data. This results in some corner cases, such as determination of the end of a message. Our proof-of-concept ends messages with a null byte (`0x00`). However, if the data being transmitted were to include `0x00` as meaningful data, this could lead to problems.

To send generic data, the byte pattern to signal the end of a message could be changed to a value that is not expected to appear in the data. TinyOS recommends sentinel bytes of `0x7D` or `0x7E` as they are the least common values in real-world empirical sensor data [11]. These bytes can then be used for framing or byte-stuffing (as done in TinyOS's Active Messages) with minimal overhead.

Another corner case is the transmission of a chunk of all `0`s. In principle, chunk lengths may range from 1-8 bits. Chunks of all `0`s are more likely to occur the smaller the chunk size is, but this does not necessarily mean an entire `0x00` byte is transmitted. Since the data to be transmitted is XOR-ed with the previous advertisement's public key to create a new public key, XOR-ing a value of all `0`s will result in the same public key. In this case, TagAlong would send out the same advertisement two rounds in a row.

Luckily, because of TagAlong's packet format, this is not an issue. Since new chunks of data are successively XOR-ed with the previous advertisement's public key, one needs to "solve" the previous advertisement's value before working on the next one. Thus decoding only queries for keys for one chunk at a time. With an all `0` chunk (at position n), even though the previous chunk's public key is the same, TagAlong is only varying one chunk at a time and will only pay attention to the nth chunk of the public key. As it queries for all 2^x possible values of the chunk (where x is the chunk bit length), TagAlong will decode all bytes including `0x00`.

4 IMPLEMENTATION

Our complete implementation of both leaf-device firmware and data recovery engine are available on GitHub at ***github.com/alex-bellon/tagalong***. We modify both the Send My firmware and the DataFetcher application to implement the TagAlong protocol.

Table 1: DRR for 1000 Messages Sent to Single Phone. We place the ESP32 in a Faraday cage with one iPhone, send 1000 unique packets 1, 2, or 5 times, and then remove the iPhone so it can upload the packets to Find My servers. This is performed five times for each configuration. The table shows how many of the 1000 unique packets made it to the Find My servers.

	Message Sent 1x	Message Sent 2x	Message Sent 5x
Trial 1	108	535	490
Trial 2	131	202	512
Trial 3	20	620	236
Trial 4	3	572	123
Trial 5	25	633	747

Data recovery begins with a standard Find My request for all the locations a device was seen at during the day. Each location is the root of a data discovery search, which iteratively searches the possible 2^n keys to see which 'device' (i.e., unique public key) was seen and reported to Apple servers in order to recover the next chunk of data. Repeating this process for each successive chunk allows the DataFetcher program to completely recover the data.

5 EXPERIMENTAL RESULTS

Does this really work? Can anyone simply deploy BLE devices with carefully crafted advertisements and recover meaningful data? We find that with modest redundancy in transmission and especially in higher-traffic areas, 'free' backhaul is readily available today.

5.1 Data Reception Rate

We evaluate reliability through data reception ratio (DRR) from ESP32 devices sending out advertisements to see if the data is received on the Find My network servers. We perform these DRR experiments by sending 1000 unique advertisements up to 5 times in different conditions; these include (1) how many times a packet is sent and (2) how many Apple devices are around for a total of 30 trials. We summarize our reliability results in Table 1 and Table 2. The reliability of TagAlong is highly dependent on the number of Apple devices in the ESP32's vicinity, but even having a single iPhone in range can provide a decent communication channel. When only one iPhone receives BLE advertisements, the DRR ranges from 0.03% to 74.7%, depending on how many times each unique packet was sent. When the BLE advertisements are broadcasted in a busy room, the DRR ranges from 53.1% all the way up to 97.2%. Given the prevalence of Apple devices in schools, office buildings, urban areas, and transportation hubs, TagAlong can be a promising communication channel.

5.2 Throughput

Under the deployment context for TagAlong, throughput is critical. Muling devices may only be in range briefly as strangers walk by devices. As such, TagAlong devices should maximize their data-per-packet. On the broadcasting side (i.e., the ESP32), the largest factor in throughput is the number of times the tweak value must be incremented before the P-224 public key encoding that encodes a part of the message becomes valid; each increment requires a new call to the cryptography library to verify whether the public key is valid or not. In our experimentation, we observed that the tweak value was never incremented more than 10 times, with most public keys only needing to be incremented 2 or 3 times. For transmit time measured from the ESP32, test messages of 10 bytes of data with a chunk size of $n = 8$ bits takes 0.8 seconds to transmit on average, which gives an output rate of 12.5 bytes/second. This only measures what can be broadcast from the ESP32, not end-to-end throughput, as many unknown and uncontrolled variables are introduced when measuring end-to-end throughput: how often nearby Apple devices are scanning for Find My packets, how long until nearby Apple devices regain an internet connection, when they uploaded their location reports, etc.

Table 2: DRR for 1000 Messages Sent in a Busy Location. We send 1000 unique packets 1, 2, or 5 times in a room with roughly 20-30 Apple devices. This is performed 5 times for each configuration. The table shows how many of the 1000 unique packets made it to the Find My servers.

	Message Sent 1x	Message Sent 2x	Message Sent 5x
Trial 1	531	739	972
Trial 2	672	644	959
Trial 3	714	709	913
Trial 4	663	832	920
Trial 5	704	738	971

Table 3: Time required to calculate candidate public keys. For each payload chunk, the DataFetcher program must calculate all possible public keys to query the Find My servers for them. The amount of time spent on this task varies by chunk length.

n (bits)	1	2	3	4	5	6	7	8
Time (ms)	4.43	9.33	17.92	34.32	65.89	131.5	287.6	511.6

The largest bottleneck in TagAlong is on the decoding side. To decode each successive chunk of data, the DataFetcher must query Find My servers for all 2^n possible values for a chunk with bit length n. To do this, the efforts on the advertising side are mirrored 2^n times by constructing the key and then incrementing the tweak value to make a valid P-224 key. The time to perform this calculation for different chunk lengths is in Table 3, measured on a 2019 MacBook Pro that runs a modified version of the TagAlong desktop application to measure only the public key calculation times. While a larger chunk size may mean more time calculating possible public keys, it also means fewer chunks to calculate. Additionally, a larger chunk size requires fewer advertisements, which is less time the mule device must be in range. The consequence is that more queries are needed to decode the entire message.

5.3 Power Consumption

As a prototype, our ESP32 implementation is not energy-optimized. However, to get a sense for the energy costs of TagAlong, we present a power profile in Figure 3 for the transmission of a 10-character message. The baseline current of the idle ESP32 is 32mA, and each large spike in current (~140-150mA) occurs when a BLE packet is transmitted. The 47mA sustained, elevated current in between the spikes is due to the ESP32 calculating the "public key" and verifying

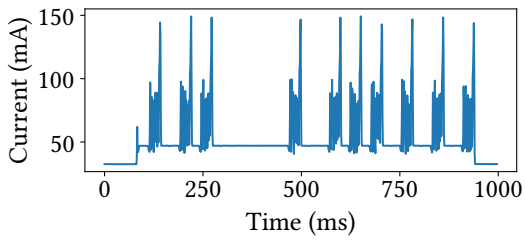

Figure 3: Current draw when broadcasting a 10-character message with chunk size of 8 (i.e. one BLE advertisement per character).

that it is indeed a valid P-224 public key. The length of the time spent calculating the keys is proportional to how many times the tweak value needs to be incremented. In this specific case, most of the public keys' tweak values only need to be incremented 1-2 times, but the 4th public key's tweak value needs to be incremented 8 times, hence the longer gap before its transmission.

6 DISCUSSION

Using Apple devices as data mules not only provides a way for commodity BLE devices to transmit data in the real world but also shows what smartphones could provide for wireless sensor nodes if they explicitly support data muling. However, while there are interesting questions and new ideas that surface from this paper, there are also ample concerns and further considerations for both Apple and BLE devices to bring this idea further to fruition.

6.1 Ethics and Concerns

TagAlong imposes an increased battery demand on third-party devices due to increased attenztion to a larger proportion of salient BLE advertisements. Additionally, the large number of 'lost' TagAlong devices will result in higher data usage and possibly higher data-plan costs for unwitting device owners.

There is also concern over how long this current faculty for third-party 'valid tags' and high-volume querying for 'lost' tags will persist before Apple imposes limitations (more realistically for the latter). If Apple revises the key generation scheme to include some form of authentication from trusted Apple silicon, then the network will no longer be accessible to commodity BLE devices, which will inhibit data exfiltration with non-Apple devices via Apple's Find My network.

6.2 Incorporating a Scanning Capability

As there is no acknowledgement channel, devices must send advertisements when there is a FindMy-capable device nearby. Prior work, such as Freeloader [1] and Positive Security's Send My [3], simply beaconed continuously, hoping that eventually a device would come by. While effective, this is very inefficient and is not well suited to energy-constrained, wireless sensor nodes.

Fortunately, Apple devices are *extremely* chatty, which allows TagAlong to efficiently detect their presence. This is primarily in support of Apple's Continuity protocol—the system designed to enable seamless handoff of interactive user context between Apple devices. Most advertise roughly three times per second [6]. If an Apple device is nearby—detected by scanning for manufacturer IDs in nearby advertisements—TagAlong can then send enough copies of its advertisements when an Apple scanner will likely hear them.

This allows for a higher probability of payload delivery, increasing with the larger number of Apple devices nearby. The downside to incorporating scanning within BLE devices is more energy consumption, considering that it is more costly to listen to advertisement packets than to send them. However, we note that this benefit and drawback depends on the type of environment devices are in too. A BLE device that advertises in an area densely populated with Apple devices might not need a scanning capability, whereas a BLE device deployed in an area with sparser human activity might benefit from scanning for Apple devices before sending data.

6.3 Location and Time Services

Location and time comes implicitly with the Find My design. Something like dedicated GPS is often too expensive in dollars or energy to justify for most sensors. However, location and time metadata is usually critical for robustness and maintenance for real-world deployments. While the location retrieved from a nearby Apple device might not accurately represent the device's exact location, there is an opportunity to achieve coarse-grained location, as opposed to no location data at all. Given BLE requires less energy than GPS, BLE could be used to obtain a initial coarse-grained estimation of location before getting more accurate measurements from GPS.

There is also an opportunity to acquire time metadata. Apple devices timestamp packets to mark when the device was seen. This could allow devices to retrieve time for wireless sensor nodes that lack a real time clock. However, this timestamp might only reflect when the data was sent, not when the sample was collected. As such, a terminator byte could be used to indicate whether a the data is fresh (i.e., sent immediately) or stale (i.e., delayed).

7 CONCLUSION

Data-muling through devices is an idea that has existed for over a decade. It has continuously been stymied by limited user uptake constraining the availability of real-world mules, however. We show that today all Apple devices can serve as data mules, which will transmit data for wireless sensor nodes at a rate of up to 12.5 $^{\text{bytes}}/_{\text{sec}}$. By tapping into the Apple infrastructure to utilize their devices and network to ferry data, we can begin to think about the potential to use the existing smartphone infrastructure as a backbone for wide-area data collection from wireless sensor nodes.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-2038238. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We thank the reviewers and our shepherd for their time and thoughtful feedback.

REFERENCES

[1] J. Adkins, B. Ghena, and P. Dutta. Freeloader's guide through the Google galaxy. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, HotMobile '19, page 111–116, New York, NY, USA, 2019. Association for Computing Machinery.

[2] Apple Platform Security. Using Find My to locate missing Apple devices. https://support.apple.com/en-gb/guide/security/sece994d0126/web, Feb 2021. Accessed: Nov 2022.

[3] F. Bräunlein. Send My: Arbitrary data transmission via Apple's Find My network. https://positive.security/blog/send-my, May 2021. Accessed: Nov 2022.

[4] A. Faggiani, E. Gregori, L. Lenzini, V. Luconi, and A. Vecchio. Smartphone-based crowdsourcing for network monitoring: Opportunities, challenges, and a case study. *IEEE Communications Magazine*, 52(1):106–113, 2014.

[5] B. R. Ghena. *Investigating Low Energy Wireless Networks for the Internet of Things*. PhD thesis, University of California, Berkeley, 12 2020. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-209.html.

[6] H. Givehchian, N. Bhaskar, E. R. Herrera, H. R. L. Soto, C. Dameff, D. Bharadia, and A. Schulman. Evaluating Physical-Layer BLE Location Tracking Attacks on Mobile Devices. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1690–1704, 2022.

[7] A. Heinrich and M. Stute. OpenHaystack. https://github.com/seemoo-lab/openhaystack, 2021. Accessed: Nov 2022; Commit: b65a6e6.

[8] A. Heinrich, M. Stute, and M. Hollick. OpenHaystack: A Framework for Tracking Personal Bluetooth Devices via Apple's Massive Find My Network. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '21, page 374–376, New York, NY, USA, 2021. Association for Computing Machinery.

[9] A. Heinrich, M. Stute, T. Kornhuber, and M. Hollick. Who Can Find My Devices? Security and Privacy of Apple's Crowd-Sourced Bluetooth Location Tracking System. *Proceedings on Privacy Enhancing Technologies*, 2021(3):227–245, 2021.

[10] U. Park and J. Heidemann. Data muling with mobile phones for sensornets. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, page 162–175, New York, NY, USA, 2011. Association for Computing Machinery.

[11] The TinyOS Developers. Serial/HDLC. http://tinyos.stanford.edu/tinyos-wiki/index.php/Serial/HDLC. Online; accessed 14-October-2022.

[12] X. Wu, K. N. Brown, and C. J. Sreenan. Analysis of smartphone user mobility traces for opportunistic data collection in wireless sensor networks. *Pervasive and Mobile Computing*, 9(6):881–891, Dec. 2013.

[13] T. Zachariah, N. Jackson, and P. Dutta. The Internet of Things Still Has a Gateway Problem. In *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications*, HotMobile'22, page 109–115, New York, NY, USA, March 2022. Association for Computing Machinery.

[14] T. Zachariah, N. Jackson, B. Ghena, and P. Dutta. ReliaBLE: Towards Reliable Communication via Bluetooth Low Energy Advertisement Networks. In *Proceedings of 2022 International Conference on Embedded Wireless Systems and Networks*, EWSN'22, 2022.

[15] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta. The Internet of Things Has a Gateway Problem. In *Proceedings of the 16th Workshop on Mobile Computing Systems and Applications*, HotMobile'15, New York, NY, USA, Feb 2015. ACM.