Key Update at Train Stations: Two-Layer Dynamic Key Update Scheme for Secure Train Communications

Sang-Yoon Chang[1,2(✉)], Shaoying Cai[3], Hwajeong Seo[3], and Yih-Chun Hu[1,4]

[1] Advanced Digital Sciences Center, Singapore, Singapore
[2] University of Colorado Colorado Springs, Colorado Springs, CO, USA
schang2@uccs.edu
[3] Institute for Infocomm Research, A*STAR, Singapore, Singapore
[4] University of Illinois at Urbana-Champaign, Urbana, IL, USA

Abstract. Modern train systems adopt communication-based train control (CBTC), which uses wireless communications to better monitor and control the train operations. Despite the well-studied security issues in wireless networking in information technology applications, security implementations in trains have been lagging; many train systems rely on security by obscurity and forgo well-established security practices such as key updates. To secure train systems against increasingly evolving and persistent attackers and mitigate key breach (which can occur due to misuse of the key), we build a key update scheme, *Key Update at Train Stations* (KUTS), that leverages the inherent physical aspects of train operations (mobility/infrastructure-asymmetry between the stations and the trains and the operational differences when the trains are at stations and between the stations). Furthermore, by incorporating separation of key chain and use and on the entities providing the key seeds, KUTS protects the key seeds for future updates against the breach of the current key and is both key-collision irrelevant (thwarting known collision-based threats on one-way random functions) and system-compromise resilient (protecting the key secrecy even when the train system is compromised). We theoretically analyze KUTS's effectiveness, security strength, and security properties. We also implement KUTS on various computing devices to study the performance overhead.

1 Introduction

Communication-based train control (CBTC) uses wireless communication to deliver operational-control messages from the train to the track-side antenna, which in turn relays the message to the operational control center (OCC) via wired connection, and vice versa. In addition to being lightweight in infrastructure and better supporting the mobility of the trains, CBTC enables finer granularity for the train vehicle's location sensing (which is a key parameter for train control) than the traditional fixed-block technology (which uses discrete railway-track segments for train localization). Consequentially, because CBTC enables

R. Deng et al. (Eds.): SecureComm 2016, LNICST 198, pp. 125–143, 2017.
DOI: 10.1007/978-3-319-59608-2_7

greater customer transport efficiency (e.g., enabling more trains to get packed per distance during busy hours), train operators have increasingly deployed CBTC for train systems.

While safety issues have been well-studied in train systems (some measures of which can also mitigate communication availability issues) because of the physical consequences of failure, security in general has not garnered much attention from the train system integrators and operators. They, instead, largely rely on security by obscurity and that the protocols are confidential and proprietary. While this does increase the barrier for security breach, especially for the type of system that is not readily accessible by the public (e.g., unlike computers or cars, not many people own/operate a train), history has shown that such approach is insufficient against motivated and persistent cyber-attackers, e.g., Stuxnet malware discovered in 2010. The security-by-obscurity approach is further challenged by the recent push to make the train systems interoperable across European nations, which will involve effort to unify and standardize the practice/design and thus make the information more obtainable [1]. Previous failures in train systems, e.g., whether accidental [2] or playful [3], demonstrate vulnerabilities in train systems and let us wonder how much more of an impact sophisticated attackers can make on train operations. Also, the recent high-profile security incidents in car applications that allowed remote (Internet-connected) attackers to take control of car operations [4–7] are alarming to train operators as well because while the wireless channels breached during these incidents deliver allegedly non-critical communications, e.g., software updates, the trains use them for critical CBTC messages that directly control the train operations. Only recently, there has been concerted effort into begin addressing security for train communication systems [8–10].

To address the security gap of train communications, we study key management within the train communication systems, which is a fundamental building block of many secure communication protocols and practice. Given an initial seed acting as a root of trust, we design a key update protocol, so that the key remains fresh and secret. Our work not only makes the key breach significantly harder but also limits the impact of such breach to the current key.

Our solution takes advantage of the unique physical aspects that are inherent in train applications. In specific, trains transport people in two phases: *at stations*, the human customers embark or get off of the train vehicle, and *between stations*, the train vehicle moves from one point to another to transport passengers and goods. Because the train at stations is static or moving much more slowly and because the station contains more infrastructure, stations provide a more tightly controlled environment for trains; CBTC implementations to track the train vehicle locations also focus more on the latter periods when the trains are travelling between the stations because such period generally presents greater challenges (less resources/equipment along the rail tracks, more exposure to the public wild, mobility of the train, and so on). We also align our scheme to these two phases and update the key when the train is at the station and use the key when the trains are between the stations, time-interleaving the key update and

its use. We thus call our scheme *key update at train stations* (KUTS); however, KUTS does not need to be implemented at all physical stations but only a subset of the stations (*KUTS stations*).

KUTS is based on one-way functions that generate pseudo-randomness. However, it defeats the known output-collision-based threats on one-way functions that significantly reduce the entropy (key strength) by introducing a two-layer approach for updating the KUTS keys; the separation between the two layers (one for the key seeds/chain and the other for the keys being used) also protect the future keys even when the currently used keys (for CBTC operations) are compromised. Furthermore, KUTS uses key seeds from both the train and the station (which are logically separate from each other) to increase resilience against train-system compromise.

The rest of the paper is organized as follows. Section 2 describes related work while Sect. 3.1 provides an overview of the train system, focusing on the part that is relevant to our work. We build the corresponding system and threat models in Sects. 3.2 and 3.3, respectively. KUTS scheme (the key update and the key failure detection) is described in Sect. 4, and we theoretically analyze its security effectiveness and properties in Sect. 5. Furthermore, KUTS is implemented and its efficiency and effectiveness analyzed in Sect. 6. Lastly, we conclude our work in Sect. 7.

2 Related Work

As discussed in Sect. 1, security developments in train networks has lagged other fields in computer security and has largely focus on wireless availability, e.g., [1,10,11]. We study an orthogonal problem of key exchange, a critical building block on which many security protocols in the digital domain rely. In related work, Lopez and Aguado [12] sketch an improvement of the European Rail Traffic Management System (ERTMS)'s outdated PKI system, which was designed in the 1990s. Separately, Hartong et al. [13] also proposes key management requirements for train systems. Our work also studies key management but, in contrast to prior work studying broader aspects of key management, we focus on improving the security by introducing updates and dynamism on the keys (with the update cycle synchronized with the physical train operations of periodic station visits).

Our work is inspired by path authentication work in computer security. Path authentication provides assurance that the object of the mechanism went through a specific path by having the relaying entities (along that path) interact with the object. It is used in the contexts of network routing [14–16] and device manufacturing/supply-chain [17–20] to identify the path and avoid the tampering of the object. Although our work is similar in the sense that the relaying nodes (stations) are stationary and help with the authentication via interactions with the moving objects (trains), KUTS is fundamentally different because the train's mobility trajectory is defined by the railway tracks and its route pre-established by the OCC while the stations have fixed geographical locations and

are interwoven with many aspects of the train operations. In other words, while the objective of path authentication in supply-chain and network-routing is to ensure that the object travels through a path, the travel path actually serves as a source of assurance in KUTS, as it is difficult to make the train diverge from the path defined by the railway tracks.

KUTS uses two layers of pseudo-random generators (hash chains). Prior work also adopt multiple layers of hashing, but the schemes are in different contexts and the different layers are for orthogonality purpose, for example, multicast source authentication work by Challal et al. [21] uses different layers for redundancy control and chooses one layer from multiple layers for execution, and Fredman et al. designed a scheme for efficient data lookup [22] that has different layers to describe orthogonal dimensions of the pointer. In contrast, KUTS uses multiple layers for greater security, and the layers are sequentially executed to generate the key update. On the other hand, our cryptographic construction shares greater similarity with Ohkubo et al.'s use of two-layer hash chains for RFID privacy [23]; their work aims to achieve forward security (preventing backward tracking, so that the breach does not enable an attacker to trace the data back through past events). However, in contrast to their work, we protect both the future and the past keys from the key breach by separating the keys being used for CBTC and the key seeds used for updates; furthermore, our construction also involves multiple independent parties distributing the key seeds to build resiliency against system compromise and is thus more complex. The use of such cryptographic constructions in resource-constrained RFID tags shows great promise that the overhead will be even more marginal for train applications, as train-borne devices has much less hardware constraints and requirements.

Our instantiation of KUTS uses SHA-256 hash for the pseudo-random generator. To put the attacker's cost in perspective, we discuss SHA-256's use in bitcoins and bitcoin mining in Sects. 4.2 and 6.2.

3 Train System Model

3.1 The Application System

Trains are designed to transport people from one geographical point to another. And to provide the customers with more options for the geographical points for their departure/arrival, trains operate on pre-established and fixed stations, which are where the customers ride or get off the train. The stations are connected with railway tracks, on which the trains operate and move, and thus the train operational trajectories are fixed/limited and clearly defined by the railway tracks. We make use of these physical aspects, e.g., the trains visit and stop at the more controlled environment of stations, to build security in the cyber-domain; for example, it is difficult for an attacker to physically take the train and have it diverge from the railway tracks.

In CBTC, the operational control center (OCC) is actively involved in the real-time control of all the trains on the line. However, since the centralized OCC controls many spatially distributed trains, it uses networking to communicate

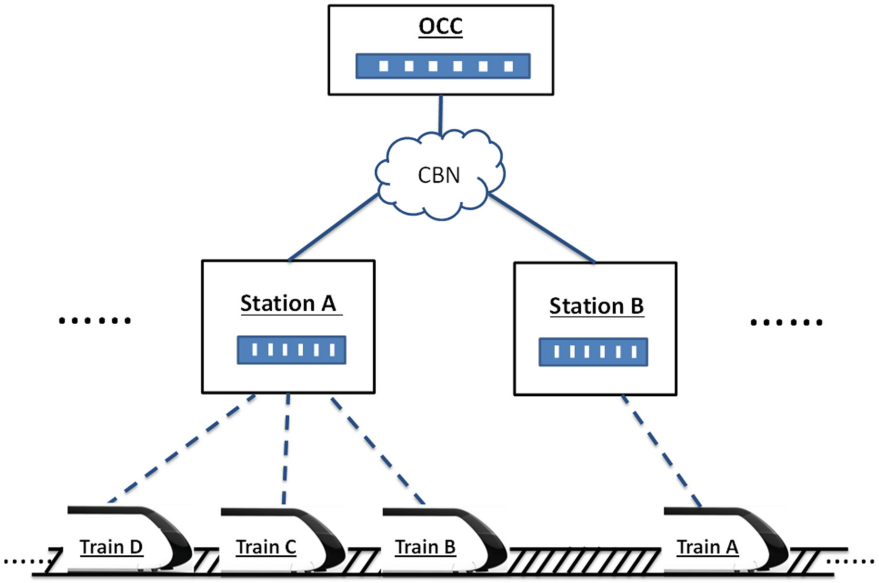


Fig. 1. CBTC communication architecture. CBN stands for communication backbone network, and the solid/dotted line represent wired/wireless connectivity, respectively.

with the trains; the OCC communicates its operational control messages, and the trains report their statuses to the OCC. OCC thus has a central view of the train line and know the time schedules/itineraries of the train operations and the train vehicles' locations at any given time, which can also be used for safety (e.g., train collision avoidance) and better traffic management. To enable real-time monitoring and control, the OCC and the trains communicate periodically; the protocol has failed if the expected messages do not arrive in a timely manner or if any of the channels conflict with each other and result in inconsistency (train systems rely on redundancy for many logical operations), which events can trigger fail-safe. Any deviation resulting in inconsistency will be detected, regardless of the failure source or the cause, and we build on such protocols to develop a failure-detection scheme in the endhosts's (OCC's and trains's) perspectives in Sect. 4.3.

Train operations rely more heavily on CBTC between stations than at stations, as discussed in Sect. 1, because of the following reasons. First, train's location (which is the most important sensing factor for CBTC) changes in faster pace than at stations, requiring greater amount of information exchange between the individual train and the OCC. Second, there are greater resources for train sensing at stations, enabling tighter and more precise control of the train operations; for example, as the train enters the stations, a dense sequential array of trackside beacons (operating independently to CBTC) are deployed for better alignment of the platform screen doors for passenger boarding.

As depicted in Fig. 1, the path between OCC and the trains are comprised of both wired connections (from the OCC to the network switches and then to the stations and/or trackside equipment) and wireless connections (between the stations/trackside and the trains); the trains physically move over long distances and thus require wireless channels. Often acting as a communication relay between the OCC and the trains, the stations also incorporate control, e.g., control booth to oversee the on-site operations.

Because the OCC primarily acts as the brain in CBTC-based train operations, it assumes the role of credential management in our work. Specifically, it allocates the identities and the roots of trust to the trains and the stations. We assume that the OCC is secure and that the initial roots of trust are established; OCC failure is beyond the scope of our contribution.

3.2 Train Model

We build our model from the unique aspects of train system operations, i.e., the hierarchical structure (from the OCC to the stations to the trains) and the mobile/static nature of trains/stations, respectively.

Given trains $i \in \mathcal{T}$ and the established railway path with the stations $j \in \mathcal{S}$ (with $\mathcal{S}$ being a vector set with the elements in a particular order), we model the train system discussed in Sect. 3.1 and the host connectivity with a graph. Figure 2 shows a sample snapshot of the connectivity graph with

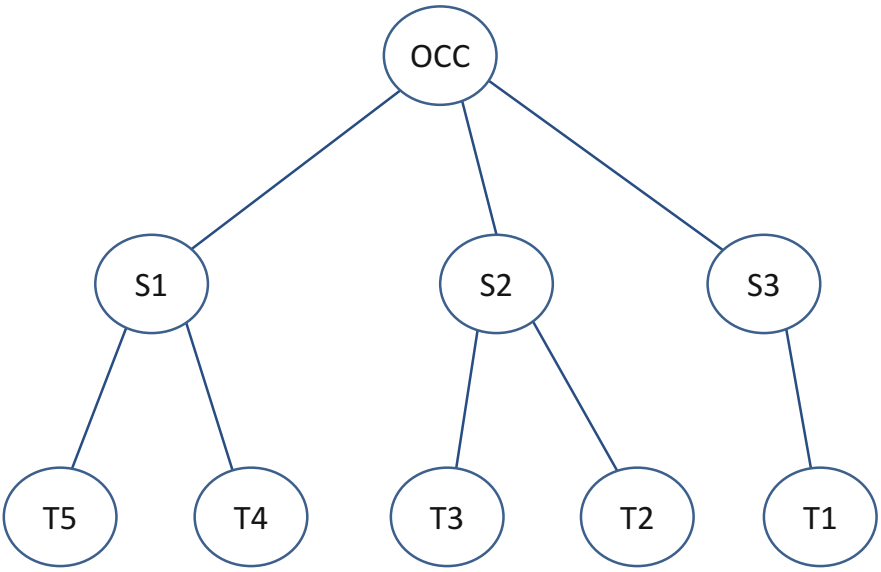


Fig. 2. A sample graph with three KUTS stations and five trains on the line. The nodes indicate the hosts involved in KUTS (with S being stations and T being trains), and the edges indicate logical KUTS interactions.

three KUTS stations and five trains on the line, i.e., $\mathcal{S} = \{S1, S2, S3\}$ and $\mathcal{T} = \{T1, T2, T3, T4, T5\}$. Only the hosts that are involved in KUTS - the OCC, KUTS stations, and trains - are represented in the graph, and not all physical train stations (where the passengers embark/debark the trains) need to be involved in KUTS[1]. Henceforth, we define *stations* to mean KUTS stations, and not the rest of the train stations irrelevant to KUTS. S nodes correspond to stations and T to trains, and the edges indicate the current connectivity. The lowest row with trains are mobile, thus making the graph dynamic in time, while the two upper rows are stationary. The indices correspond to time where the trains travel from left to right. For example, T1 is the first train, T2 the second train, and so on. Similarly, S1 is the first station that the trains encounter, S2 the second, and so on. In the figure, for example, T1 passed S1 and S2 and is currently either stationed at S3 or just passed S3 enroute to the next station. Our model applies generally to the train-line topology, e.g., for a line that is a loop with 5 stations, the same physical station can be captured by incrementing j for every train cycle (so that the physical station owns all the indices of j modulo 5 and enables that the station's keys be unique per train cycle) and can be straightforwardly adapted for multiple train-route cases (as long as there is a countably finite sets of stations and trains and the OCC knows each of the train's routes); in fact, KUTS is largely described in each of the train's view in Sect. 4, e.g., station j corresponds to the j-th station that a train encounters.

As described in Sect. 3.1, the OCC distributes keys to the stations and the trains, and the trains use those keys to secure their communications to/from the stations and the OCC. The stations' key seeds are denoted k^j while the trains' key seeds are k^i_j where j and i are, respectively, the station index and train index as described previously; the presence of the subscript indicates whether it is train's key seed (subscript present) or station's key seed (subscript absent). The subscript for train key seeds corresponds to time and gets incremented when the train passes KUTS stations, i.e., j in k^i_j corresponds to the last station that the train stopped. For example, for Fig. 2, T1 currently passed S3 and thus uses k^1_3 to drive the key chain. As the initial root-of-trust, the OCC also distributes k^i_0 to train i. KUTS provides a key update scheme given k^j, $\forall j \in \mathcal{S}$ and k^i_0, $\forall i \in \mathcal{T}$. In other words, given a train i and its initial seed k^i_0, KUTS computes k^i_j (when passing the station j), which afterward is used to generate the keys that are used for securing train communication protocols. Section 4 discusses KUTS in greater details.

3.3 Threat Model

As in other key-establishment work, we consider attackers whose objectives are to learn the key. In addition to the traditional threat model of attackers residing outside of the train system, we consider any attackers but the OCC (the

[1] The choice of KUTS stations is a design parameter which has a tradeoff between security strength and complexity/computation and is beyond the scope of this paper. Section 5 provides analyses and insights that can be helpful in making such design choices.

trusted authority, as discussed in Sect. 3.1) and the corresponding train (who owns the set of keys). The attackers do not have control over or did not compromise the hosts depicted in Fig. 2 (the logical entities that govern KUTS), but they can physically reside within the train infrastructure system, e.g., the station-trackside relay/switch or the parts of the station irrelevant to KUTS intelligence are within the scope of our attacker model. Such attackers can also conduct active attacks and disrupt the KUTS by diverging from the protocol, e.g., drop the KUTS exchange or relay incorrect keys; we develop a detection countermeasure for such active threats in Sect. 4.3. Such insider compromise (where attackers breached parts of the system) is increasingly being considered in critical infrastructure security, such as in the car vehicular networking (multiple credential authorities collaborating with each other for vehicular credential management system) [24] and in device and chip manufacturing (split manufacturing) [25,26].

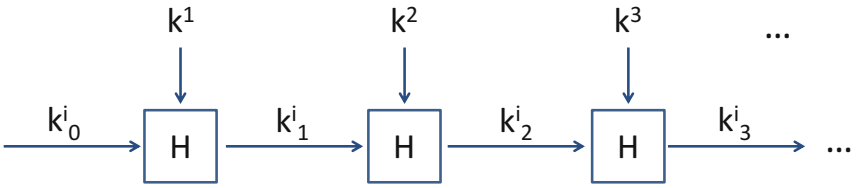
4 KUTS Scheme

We build our scheme on the model described in Sect. 3.2 and, using the jth station's key seed (k^j) and train i's initial key seed (k_0^i), describe KUTS which updates the key for dynamic key establishment. The key updates are generated on the train when it is at the KUTS stations (OCC, keeping track of the trains' locations, also separately update the key using KUTS) and used while the train is moving between the stations, which provides well-defined time-separation between the key updates and use. We use well-established cryptographic tools, e.g., one-way functions, for the key update in Sect. 4.2 and describe the key failure detection in Sect. 4.3. But first, we define the contribution scope of KUTS in Sect. 4.1.

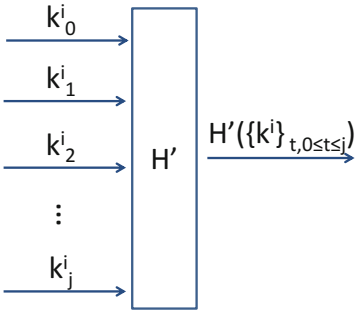
4.1 KUTS Contribution Scope

Our contribution lies in establishing the keys between the OCC and trains for CBTC, but not in *how* to use those keys to secure the communications. How to use the keys for secure networking depends on the threat model and the corresponding threat vectors, on which the security measure focuses, and such developments are widely studied in computer security. For example, the keys can generate digital signatures for communication integrity; the keys can be used for message encryption for confidentiality; the keys can drive randomization to thwart network reconnaissance or wireless denial-of-service attacks; and so on. Our work thus serves as a building block to secure communication against attackers in various scopes (whether they compromised the network and are physically residing within the train network, e.g., switch, or have access to the wireless link between the train and the station). Our analyses for KUTS in Sect. 5 also supports such generality in key use and attacker scope.

We focus on key updates given a key infrastructure with key seeds distributed a priori, as discussed in Sect. 3.2. While key updates in other contexts, such



(a) KUTS key-chain update



(b) KUTS key generation

Fig. 3. KUTS scheme for train i

as those discussed in Sect. 2, are either time-dependent (periodic updates with regular time intervals) or event-based (triggered by an event), KUTS update is space-dependent (updates at the stations) and uses the train's pre-established route (defined by the railway tracks and publicly announced).

4.2 KUTS Update and Key Generation

OCC, as the trusted authority, knows all the keys. For any $i \in \mathcal{T}$ and $j \in \mathcal{S}$, the station j knows its own key seed k^j and shares it to train i upon the arrival of the expected train i. Train i takes its current key seed k^i_{j-1} and the station's key seed k^j to update its key seed to k^i_j. Only k^j is communicated, and the other keys (e.g., k^i_{j-1} and its history) are locally stored within the train and the OCC; the computations are also performed locally. We design KUTS to provide such an update and, for scalability, keep the train's key seeds k^i_j, $\forall j \in \mathcal{S}$ the same size.

KUTS uses two cryptographic one-way functions, H and H'. H is used for key seed update and to drive the one-way chain, while H' is used for generating the key that will be actually used for CBTC. In other words,

$$H : (k^i_{j-1}, k^j) \rightarrow k^i_j \tag{1}$$

$$H' : \{k^i_t\}_{0 \le t \le j} \rightarrow H'(\{k^i_t\}_{0 \le t \le j}) \tag{2}$$

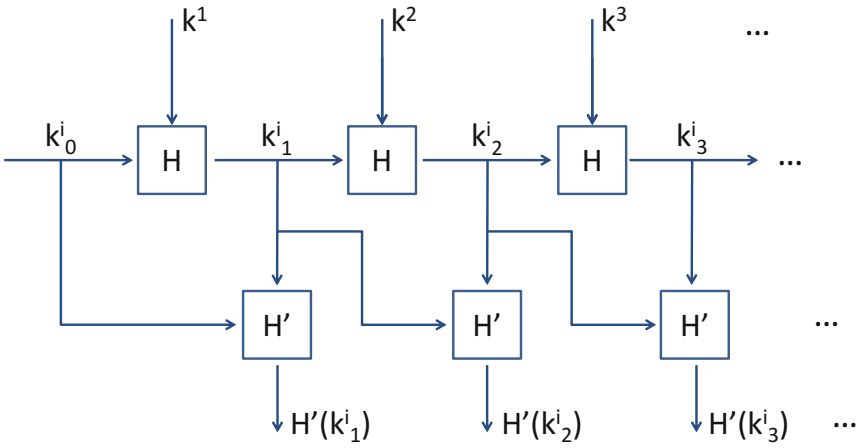


Fig. 4. Our KUTS instantiation

As cryptographic one-way functions, both H and H' are easy to compute but difficult to reverse, i.e., given $H(x)$ for some x, it is difficult to find that x. Also, as discussed in Sect. 5.1, KUTS is not sensitive to collision and is not subject to many collision-based attacks on one-way computations.

Figure 3(a) describes KUTS's key seed updates using H. For any $i \in \mathcal{T}$ and $j \in \mathcal{S}$, train i receives k^j when it is stopped at station j (the upper row of key seeds), then it uses H to compute k^i_j from k^j and k^i_{j-1} (the lower row of key seeds). Afterward, as described in Fig. 3(b), it uses $\{k^i_t\}_{0\le t\le j}$ to compute $H'(\{k^i_t\}_{0\le t\le j})$; the weight can be controlled for the input seeds of $\{k^i_t\}_{0\le t\le j}$; for example, our instantiation described in Fig. 4 uses only k^i_j and k^i_{j-1} at station j. $H'(\{k^i_t\}_{0\le t\le j})$ is used as a key for secure CBTC operations, i.e., k^j_i are not designed to be directly used. Introducing additional complexity of H' protects the security of the key chain, as discussed in Sect. 5.1.

We build an instantiation of KUTS, described in Fig. 4, to analyze the efficiency and the overhead in Sect. 6. In particular, we use SHA-256 hashes for both H and H' with input size of 512 bits. We use SHA-256 because it is quick (as we will see in Sect. 6.1) and is computationally infeasible to reverse the computation. Since we use a 512-bit-long inputs for H and H', the input for H is a concatenation of k^j and k^i_{j-1}, and the input for H' is a concatenation of k^i_{j-1} and k^i_j.

Like KUTS, bit-coins use one-way hash chains driven by SHA-256. However, KUTS is fundamentally different from bit-coin mining in the following three aspects. First, the upcoming bit-coin blocks are generated by solving the reverse of Hash or finding collisions, while KUTS is given the inputs and compute the Hash in the forward-direction to generate the key updates. Second, while the bit-coin hash blocks, once solved, are publicly advertised, KUTS-output keys are sensitive and are designed to be protected from adversaries. Third, a realistic adversary (even

for a particularly persistent one) against train systems is significantly less computationally capable than the combined effort of bit-coin miners, as bit-coin miners are a global network of machines with some individual miners investing millions of dollars (even with such massive-scale distributed network, each SHA-256-based bit-coin block gets solved roughly in every ten minutes by design).

4.3 Detection of Key Failure

KUTS key update *failure* occurs when the OCC and the train do not agree on the key, and the failure event can happen because of the fault on the train side (e.g., receiving the station's key seeds and computing the updates) or the fault on the station side (e.g., transferring the station's seeds to the trains). KUTS detection uses the following aspects of the train communication infrastructure: it is hierarchical with OCC communicating with the stations and the stations communicating with the trains and vice versa, as described in Fig. 2; it is also dynamic with the trains moving from one station to another; and the trains' operations (and their KUTS key uses) are orthogonal to each other.

KUTS detects key-update failures (e.g., an outdated/incorrect key is used) when two or more distinct failure events are observed, where the *distinctness* of failure events come from different connectivity edges (relayed from different stations in the OCC view) or from the same edge but with different train/time instances (where time instances are according to j updates in any of the train's view, e.g., any train on the station's child branch leaves or enters). The distinctness is required because the key update relies on the cooperation with the stations and needs to distinguish between the protocol failure on the station and that on the train. This limits the failure event to one occurrence until another distinct failure event is observed, which can trigger further investigation on the node that has repeatedly caused failures. In the case of an update failure event (and before the second event occurs), the train involved in the failure uses the key before the failed update, and proceeds with the key for KUTS once it arrives the next station; OCC keeps track of this to leave a record of key failures.

For example, in Fig. 2, suppose the OCC senses a KUTS failure event when the communication got relayed from S2 and originated from T4. However, since it does not know whether the failure is from S2 or T4, it waits until T4 moves to S3 or until T5 enters S2. If T4 is misbehaving, then a failure event will occur on S3 or any other future stations that T4 will encounter; if S2 is the one misbehaving, then another failure event will occur for T5 or any of the later trains. After the first observation of the failure (with k_2^4), T4 and the OCC use the key that they agreed on before T4 arrived at S2 (k_1^4) before arriving to S3 and, upon arriving in S3, updates the new key with $H(k_1^4, k^3)$.

5 Security Analyses

We analyze the KUTS scheme in this section. While Sect. 5.1 derives security properties from the KUTS design, Sect. 5.2 theoretically analyzes the security strength.

5.1 Security Properties

KUTS is designed carefully to have the following properties that will be useful in securing the train communications. In addition to being scalable (because the key sizes remain the same for all updates) and enabling detection of misbehaving insider station or train (as described in Sect. 4.3), it is interwound with the established physical operations of the trains, insensitive to key collisions (defending some known attacks on hash algorithms), and robust to compromise. We discuss about these properties in greater details in this section.

Established and Publicly Known Trajectory of Train Operations. KUTS uses the established operational trajectory/path of the trains as a source of security assurance. Because the path is clearly defined by the railway tracks and many users involved in the operation (the train-borne customers, the train-borne logic, the stations, the OCC, and so on) a priori agreed on the path, it is difficult to change the train's operational path during CBTC (i.e., when KUTS keys are used). For example, in contrast to the path authentication work in packet routing and supply chains discussed in Sect. 2, it is challenging to make the train physically diverge from the railway tracks, re-route it without any of the entities noticing, or stop it in between stations and engage the train with KUTS; not only do the trains themselves also keep track of their own locations relative to the stations, e.g., by using odometry and/or beacons, but there are also additional redundant mechanisms to check whether the train is at a station, e.g., more capable and densely populated sensors enable higher precisions on train sensing and better alignment with the platform screen doors.

Two-Layer Approach for Collision Irrelevance. KUTS introducing two distinct hash computations (H using the key seeds and H' using the outputs of H) provides the following security properties. First, collision-based attacks on KUTS-output keys do not breach the security of the KUTS key chain because of the separation of the two hashes, as finding a collision of one hash does not also yield a collision in the other hash. As it is generally easier to find a collision than the exact hash input, this property makes KUTS more secure and thwarts many collision-based threats that has been studied to break hash algorithms. Thus, for many one-way functions, e.g., SHA-256 hash that we used in our instantiation implementation, the state-of-the-art attackers are forced to resort to brute-force. Second, it enables the additional protection of key seeds, as the key seeds are separate from H' and are not directly used to generate the keys that are actually being used for train communications. We investigate this further in Sect. 5.2.

Two-Seed Approach for Compromise Resilience. In addition to separating the key chain from the CBTC-driving keys, the train's key seed and the station's key seed are independent and originate from separate entities. Therefore, even if either of them gets compromised (which by itself is a difficult task, as the train key seed is stored and computed inside the train vehicle with no need for networking, and there is a mature set of digital cryptography techniques that

can be used to ensure confidentiality of the train-station exchange for the station's seed, e.g., using the current before-update key), the entropy for the other key seed still holds against the attacker. Section 5.2 investigates this property further against varying attacker capabilities.

5.2 Security Strength Analyses

We analyze the security strength of KUTS, against an attacker whose goal is to learn the key as described in Sect. 3.3, and use the metric of entropy which quantifies how random the value is against an unauthorized attacker [27,28] to abstract away from the key length and other implementation details (and the corresponding information leakage). The *entropy* of a discrete random value α, whose sample size is $\mathcal{S}$, is $\mathcal{H}(\alpha)$, and $\mathcal{H}(\alpha) = -\sum_{i \in \mathcal{S}} \Pr_i \log(\Pr_i)$ where $\Pr_i$ is the probability of i occurring; if the logarithm is base-2, then $\mathcal{H}(\alpha)$ is in bits. The entropy $\mathcal{H}$ becomes additive across independent random values. For example, if α is a sequence of independent uniformly-distributed bits (standard practice for digital keys) that are n bits long, then $\Pr[\alpha = \gamma] = \frac{1}{2^{\mathcal{H}(\alpha)}} = \frac{1}{2^n}$, $\forall \gamma \in \mathcal{S}$, and it takes the attacker $2^{\mathcal{H}(\alpha)-1} = 2^{n-1}$ trials to guess the correct α in expectation.

Definition 1. Given any function f, y_f is the entropy of the output of f, and x_f is the entropy of the input of f. In other words, if $\beta = f(\alpha)$, $y_f = \mathcal{H}(\beta)$ and $x_f = \mathcal{H}(\alpha)$.

In our instantiation, $x_f > y_f$ for both $f = H$ and $f = H'$, because both H and H' compress information (lossy) and have longer inputs than outputs. We initially assume that the keys, both k^i_j and k^j, $\forall j \in \mathcal{S}$, $\forall i \in \mathcal{T}$, are secure. However, we take a step-by-step approach to introduce stronger threat scenarios (under our threat model in Sect. 3.3) to break the assumption and show that KUTS still remains secure and the key random; the rest of the section is organized in the increasing order of attacker capability/difficulty. Through this analysis, we highlight the effectiveness of the separation and provide insights helpful for choosing the parameters for KUTS, such as the key seed length.

General Security Strength of KUTS. For static keys, the attacker only needs to breach the key that is being used for the train networking, and the entropy is $y_{H'}$.

In contrast, KUTS dynamically updates the keys. Because it separates the key chain and the key used for CBTC, as described in Sect. 5.1, and the computations are done locally within the KUTS machines, the attacker needs to jointly attack H and H' (as H is beyond H'), and the cost of doing so is x_H. For example, against our implementation instantiation described in Fig. 4, $x_H = \mathcal{H}(k^i_{j-1}, k^j) = \mathcal{H}(k^i_{j-1}) + \mathcal{H}(k^j)$ where the last equality comes from k^i_{j-1} and k^j being independent to each other. KUTS key chain remains secure if x_H is positive, because the *exact* input is required and KUTS is irrelevant to hash collisions; in fact, the following paragraphs study when parts of KUTS is compromised, starting from when H' is breached and $y_{H'} = 0$.

Outsider Attacker Breaching H'
Depending on the use of the KUTS-driven CBTC keys (H') and the computational capability of an attacker, the attacker can learn H'. Suppose this happens and $y_{H'} = 0$. If static keys were used, the key is compromised, and the attacker has access to the CBTC communications.

On the other hand, against KUTS dynamic keys, the attack is mitigated and its breach impact is limited to the following key update, as the rest of the KUTS is still secure, e.g., $x_{H'} = \mathcal{H}(k_j^i, k^j) > 0$ for our instantiation, because H' is a one-way function and is not injective, i.e., many-to-one mapping. Section 4.3 also provides a fall-back mechanism against temporarily compromised keys.

Infrastructure Compromise and Breaching k^j
An insider attacker who compromised the train infrastructure can learn k^j, for example, by attacking the key exchange between the station-side KUTS machine and the train. In cases where a capable insider attacker breaches the key k^j, i.e., $\mathcal{H}(k^j) = 0$, KUTS still remains secure because of the randomness in k_{j-1}^i and $x_H = \mathcal{H}(k_{j-1}^i) > 0$.

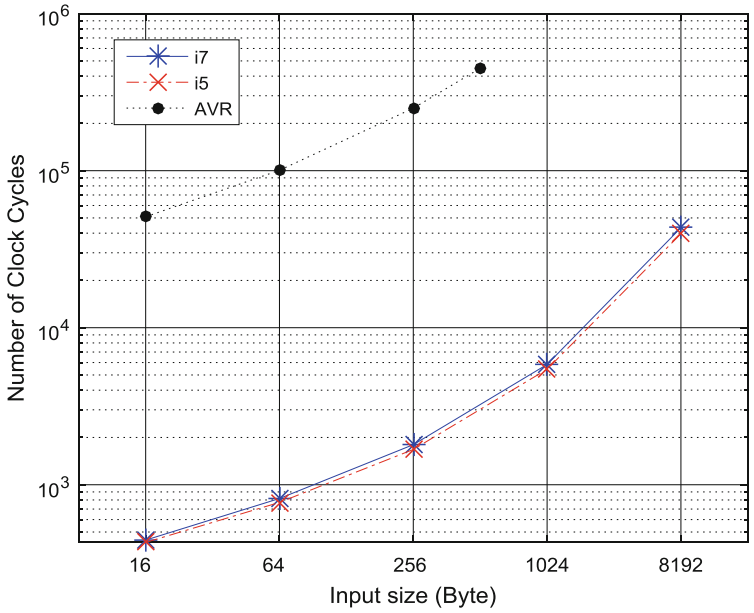
Train Compromise and Breaching k_{j-1}^i
It is difficult to compromise k_{j-1}^i because this key does not leave the train's KUTS engine, which also performs the local computations for KUTS updates. Nevertheless, even if k_{j-1}^i is compromised and $\mathcal{H}(k_{j-1}^i) = 0$, $x_H = \mathcal{H}(k^j) > 0$ and the security strength depends on k^j.

6 Implementation Analyses

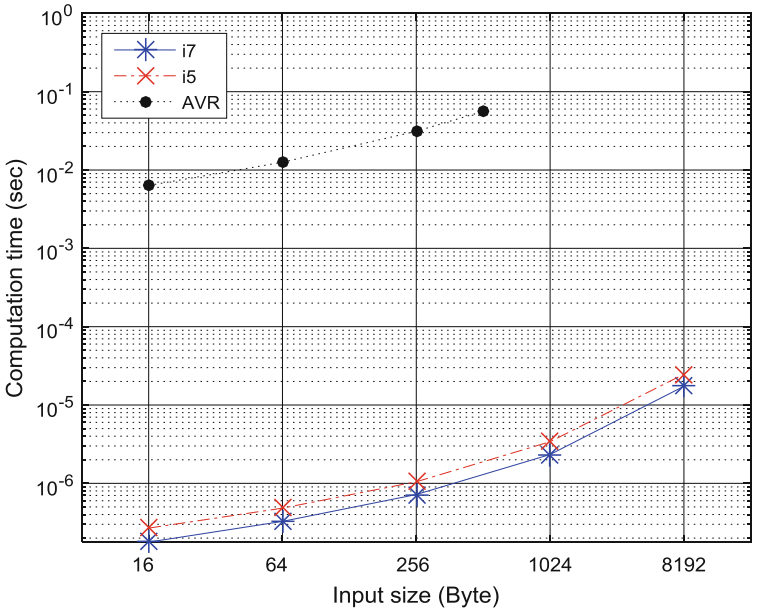
To estimate the computational overhead, we implement KUTS on three machines: *i7* (Intel i7 64-bit processor at 2.5 GHz, 16 GB RAM, Mac OS), *i5* (Intel i5 64-bit processor at 1.6 GHz, 16 GB RAM, Linux OS), and *AVR* (Atmel AVR 8-bit microcontroller at 8 MHz, 4 KB RAM). i7 and i5 machines are widely used for general-purpose computers, and AVR microcontrollers are playing greater roles in modern computing as more devices and applications, especially those constrained in resource, require logic and connectivity to realize Internet of Things (IoT); even with the processing-limited AVR and no effort to optimize, KUTS overhead is marginal as discussed in Sect. 6.2. SHA-256-based hash algorithms (for KUTS H and H') are adapted from OpenSSL, which provides a commercial-grade open-source library, widely adopted on modern-day Internet and other digital transactions.

6.1 Computational Overhead from Hashing

Figure 5(a) and (b) show the average processing overhead of computing a hash, respectively, in clock cycles and in seconds while varying the hash input size. Both i7 and i5 use 64-bit processors and have comparable processing overhead in clock cycles (e.g., i5 is slightly more efficient, requiring 3–10% less number of



(a) Computation overhead in clock cycles



(b) Computation overhead in time

Fig. 5. KUTS hash computation overhead

clock cycles to compute a hash depending on the input size), and the primary difference between i7 and i5 in seconds is derived from the clock frequency. On the other hand, the AVR significantly requires greater overhead in both clock cycles and seconds; it requires 118–150 times more clock cycles than i5, depending on the hash input size, and 114–139 times greater than i7. The difference becomes even greater in seconds due to the processing frequency difference; AVR at least takes 3.578×10^4 times longer than i7, and the difference in computational time becomes greater than four orders of magnitude. We leverage these measurements to estimate the cost of KUTS in Sect. 6.2.

6.2 KUTS Cost Analysis Between Train and Attacker

As described in Sect. 4.2, to implement KUTS, we use SHA-256 for both H and H', and the input for H is a concatenation of k^j and k^i_{j-1} while the input for H' is a concatenation of k^i_{j-1} and k^i_j. Thus, we focus on 512-bit or 64-Byte inputs. For KUTS, the overhead is dominated by the two hash computations of H and H' (the overhead from memory-based processing, i.e., read/write of the bit registry, is relatively minor). For a legitimate train, the computation takes $2 \times 326.9 = 653.8$ ns for i7, $2 \times 482.4 = 964.8$ ns for i5, and $2 \times 12.57 = 25.14$ ms for AVR. Even if the train system requires the security measure to be entirely modular to the rest of the system and uses the AVR microcontroller, the KUTS processing time overhead (of less than 3% of a second) is dominated by the time spent at the trains with customer and physical-operation-related delays (which are in the order of seconds) and is thus acceptable for deployment.

On the other hand, the attacker cannot access the hash chain and, not having the inputs of the hash, need to resort to brute-force, as described in Sect. 5.1. For example, in our implementation using the most capable i7, the attacker cost to break KUTS is $2 \times 2^{512-1} \times 326.9$ ns $= 1.409 \times 10^{140}$ years in expectation. In contrast, depending on the design of the train operations, our key updates can occur in the order of minutes (for urban metros) or hours (for rural inter-city trains). Our result corroborates with the general belief that SHA is secure enough (KUTS is also insensitive to collisions) and is thus widely used for security-sensitive digital transactions such as finance and crypto-currency (as discussed in Sect. 4.2); it will take a computing power as big as the globally distributed network of bit-coin miners to compete with the KUTS updates.

7 Conclusion

To achieve secure key establishment for train-to-infrastructure networking, we develop a key update scheme KUTS that mitigates the key breach by limiting the breach impact to the current key and builds resiliency against system compromise. KUTS design is tightly interwound with the inherent train applications (the hierarchical architecture for the vehicle-to-infrastructure CBTC communications and the differences in physical operations at and between stations, which lead to the logical separation of KUTS key update and use). We also provide

a temporary fall-back mechanism and a detection scheme, which can be used to trigger a failure-response mechanism. We build KUTS based on the state-of-the-art pseudo-random generator function (SHA-256 in our instantiation) and analyze its security strength and properties while keeping the security overhead marginal (a small fraction of a second per KUTS operation).

Acknowledgments. This work was supported by the National Research Foundation (NRF), Prime Ministers Office, Singapore, under its National Cybersecurity R&D Programme (Award No. NRF2014NCR-NCR001-31) and administered by the National Cybersecurity R&D Directorate and by the Human-Centered Cyber-physical Systems Programme at the Advanced Digital Sciences Center from Singapore's Agency for Science, Technology and Research (A*STAR).

References

1. Heddebaut, M., Mili, S., Sodoyer, D., Jacob, E., Aguado, M., Zamalloa, C.P., Lopez, I., Deniau, V.: Towards a resilient railway communication network against electromagnetic attacks. In: TRA - Transport Research Arena, France, p. 10p, April 2014. https://hal.archives-ouvertes.fr/hal-01061258
2. He, H.: Passenger wi-fi freezes third Shenzhen metro train in a week. South China Morning Post. http://www.scmp.com/news/china/article/1078165/passenger-wi-fi-freezes-third-shenzhen-metro-train-week
3. Squatriglia, C.: Polish teen hacks his citys trams, chaos ensues. Wired. http://www.wired.com/2008/01/polish-teen-hac/
4. Greenber, A.: Hackers remotely kill a jeep on the highwaywith me in it. Wired. http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/
5. Finkle, J., Woodall, B.: Researcher says can hack GM's OnStar app, open vehicle, start engine, Reuters. http://www.reuters.com/article/2015/07/30/us-gm-hacking-idUSKCN0Q42FI20150730
6. Foster, I., Prudhomme, A., Koscher, K., Savage, S.: Fast and vulnerable: a story of telematic failures. In: 9th USENIX Workshop on Offensive Technologies (WOOT 2015). USENIX Association, Washington, D.C., August 2015. http://blogs.usenix.org/conference/woot15/workshop-program/presentation/foster
7. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T.: Comprehensive experimental analyses of automotive attack surfaces. In: Proceedings of the 20th USENIX Conference on Security, SEC 2011, p. 6. USENIX Association, Berkeley (2011). http://dl.acm.org/citation.cfm?id=2028067.2028073
8. American Public Transportation Association (APTA): Securing control and communications systems in rail transmit environments, part II: Defining a security zone architecture for rail transit and protecting critical zones, Recommended Practice, ATPA-SS-CCS-RP-002-13 (2013)
9. American Public Transportation Association (APTA): Cybersecurity considerations for public transit, Recommended Practice, ATPA-SS-ECS-RP-001-14 (2014)
10. Deniau, V.: Overview of the European project security of railways in Europe against electromagnetic attacks (secret). IEEE Electromagn. Compat. Mag. **3**(4), 80–85 (2014)

11. Chang, S.-Y., Tran, B.A.N., Hu, Y.-C., Jones, D.L.: Jamming with power boost: leaky waveguide vulnerability in train systems. In: 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), pp. 37–43, December 2015
12. Lopez, I., Aguado, M.: Cyber security analysis of the european train control system. IEEE Commun. Mag. **53**(10), 110–116 (2015)
13. Hartong, M., Goel, R., Wijesekera, D.: Key management requirements for positive train control communications security. In: Proceedings of the 2006 IEEE/ASME Joint Rail Conference, pp. 253–262, April 2006
14. Reiter, M., Stubblebine, S.: Resilient authentication using path independence. IEEE Trans. Comput. **47**(12), 1351–1362 (1998)
15. Zhao, M., Smith, S.W., Nicol, D.M.: Aggregated path authentication for efficient BGP security. In: Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, pp. 128–138. ACM, New York (2005). http://doi.acm.org/10.1145/1102120.1102139
16. Kim, T.H.-J., Basescu, C., Jia, L., Lee, S.B., Hu, Y.-C., Perrig, A.: Lightweight source authentication and path validation. In: Proceedings of the 2014 ACM Conference on SIGCOMM, SIGCOMM 2014, pp. 271–282. ACM, New York (2014). http://doi.acm.org/10.1145/2619239.2626323
17. Blass, E.-O., Elkhiyaoui, K., Molva, R.: Tracker: security and privacy for RFID-based supply chains. In: 18th Annual Network and Distributed System Security Symposium NDSS 2011, 6–9 February 2011, San Diego, CA, USA (2011). http://www.eurecom.fr/publication/3233. Accessed Feb 2011
18. Elkhiyaoui, K., Blass, E.-O., Molva, R.: CHECKER: on-site checking in RFID-based supply chains. In: Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks, WISEC 2012, pp. 173–184. ACM, New York (2012). http://doi.acm.org/10.1145/2185448.2185471
19. Cai, S., Li, Y., Zhao, Y.: Distributed path authentication for dynamic RFID-enabled supply chains. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) Information Security and Privacy Research. SEC 2012. IFIPAICT, vol. 376, pp. 501–512. Springer, Heidelberg (2012)
20. Cai, S., Deng, R.H., Li, Y., Zhao, Y.: A new framework for privacy of RFID path authentication. In: Bao, F., Samarati, P., Zhou, J. (eds.) ACNS 2012. LNCS, vol. 7341, pp. 473–488. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31284-7_28
21. Challal, Y., Bouabdallah, A., Hinard, Y.: Efficient multicast source authentication using layered hash-chaining scheme. In: 29th Annual IEEE International Conference on Local Computer Networks, pp. 411–412, November 2004
22. Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with 0(1) worst case access time. J. ACM **31**(3), 538–544 (1984). http://doi.acm.org/10.1145/828.1884
23. Ohkubo, M., Suzuki, K., Kinoshita, S.: Cryptographic approach to "privacy-friendly" tags. In: RFID Privacy Workshop (2003)
24. Whyte, W., Weimerskirch, A., Kumar, V., Hehn, T.: A security credential management system for V2V communications. In: Vehicular Networking Conference (VNC), pp. 1–8. IEEE, December 2013
25. Rajendran, J., Sinanoglu, O., Karri, R.: Is split manufacturing secure? In: Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1259–1264, March 2013

26. Imeson, F., Emtenan, A., Garg, S., Tripunitara, M.: Securing computer hardware using 3D integrated circuit (IC) technology, split manufacturing for obfuscation. In: Presented as Part of the 22nd USENIX Security Symposium (USENIX Security 2013), pp. 495–510. USENIX, Washington, D.C. (2013). https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/imeson
27. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**(3), 379–423 (1948). http://dx.doi.org/10.1002/j.1538-7305.1948.tb01338.x
28. Shannon, C.E.: Communication theory of secrecy systems. Bell Syst. Tech. J. **28**(4), 656–715 (1949)