CEIVE: Combating Caller ID Spoofing on 4G Mobile Phones Via Callee-Only Inference and Verification

Haotian Deng
Purdue University

Weicheng Wang
Purdue University

Chunyi Peng
Purdue University

ABSTRACT

Caller ID spoofing forges the authentic caller identity, thus making the call appear to originate from another user. This seemingly simple attack technique has been used in the growing telephony frauds and scam calls, resulting in substantial monetary loss and victim complaints. Unfortunately, caller ID spoofing is easy to launch, yet hard to defend; no effective and practical defense solutions are in place to date.

In this paper, we propose CEIVE (**C**all**e**e-only **i**nference and **ve**rification), an effective and practical defense against caller ID spoofing. It is a victim callee only solution without requiring additional infrastructure support or changes on telephony systems. We formulate the design as an inference and verification problem. Given an incoming call, CEIVE leverages a callback session and its associated call signaling observed at the phone to infer the call state of the other party. It further compares with the anticipated call state, thus quickly verifying whether the incoming call comes from the originating number. We exploit the standardized call signaling messages to extract useful features, and devise call-specific verification and learning to handle diversity and extensibility. We implement CEIVE on Android phones and test it with all top four US mobile carriers, one landline and two small carriers. It shows 100% accuracy in almost all tested spoofing scenarios except one special, targeted attack case.

CCS CONCEPTS

• **Security and privacy** → **Spoofing attacks**; • **Networks** → **Mobile networks**; **Signaling protocols**;

KEYWORDS

Caller ID spoofing; Callee-only defense; 4G Signaling; CEIVE

ACM ISBN 978-1-4503-5903-0/18/10…$15.00
https://doi.org/10.1145/3241539.3241573

ACM Reference Format:
Haotian Deng, Weicheng Wang, and Chunyi Peng. 2018. CEIVE: Combating Caller ID Spoofing on 4G Mobile Phones Via Callee-Only Inference and Verification. In *The 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18), October 29-November 2, 2018, New Delhi, India.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3241539.3241573

1 INTRODUCTION

Voice call has been a killer communication service for mobile users for decades. In recent years, despite the various security mechanisms deployed inside the carrier infrastructure and the device OS, a substantial number of telephony frauds, including scam calls, spam calls, and voice phishing, have been reported [44]. The victim leaks confidential information to the attacker during the call, resulting in business, property, or monetary losses. Even worse, the number of victims suffered from telephony frauds are growing at an alarming rate. Scam calls have been regularly reported (e.g., [2, 6, 21, 26, 34, 40, 42, 49]), and imposter scam has been the No.2 source of consumer complaints according to FTC report in 2017 [25]. An estimated one in every 10 American adults lost money in a phone scam in the past 12 months with $430 loss on average, totaling about $9.5 billion overall in 2017 [37]; These scams went up nearly 60% and the average loss increased by 56% ($274 in 2016) from a year ago. Similar losses and complaints are reported in Europe, Asia, Australia and globally [7, 8, 11, 20, 23, 55].

A simple, yet menacing attack technique behind telephony frauds is through caller ID spoofing. The attacker acts as the caller, and spoofs its caller ID (i.e., the caller name or phone number or other identities). Upon receiving the call, the victim is deceived to believe that the call comes from the "*trusted*" caller indicated by the spoofed ID (e.g., government agencies, public and utility services, banks, insurances, etc).

Specifically, caller ID spoofing uses two means to deceive the victims. The first is to simply spoof the caller name by claiming to be the trusted party (e.g., an IRS agent, Microsoft technical support) whereas the originating phone number is not spoofed (not from IRS or Microsoft). This is relatively easier to resolve given the increasing usage of phone number search service [5, 45, 50, 51, 53]. By leveraging online public information and crowdsourcing records, we can verify the calling party (i.e., whether the call is indeed from IRS or Microsoft). The second is more difficult to defend. The attacker forges the phone number of the trusted caller so that the call appears to come from the "*correct*" number of the authentic party. The solution based on phone number search thus does not work. In fact, no simple and effective solution is available for practical defense. This is the focus of this work.

In this paper, we first empirically confirm that, caller ID spoofing is indeed easy to launch, but hard to defend. Existing solution proposals are deemed ineffective due to heavy deployment and major updates on the telephony systems. These include the approaches of building a global certificate authority for end-to-end caller authentication [22, 24, 27, 35, 52], enabling network assistance on caller verification [46], launching challenge-and-response to verify the true caller (changes required on all possible callers) [38, 39], etc..

Instead, we propose **CEIVE** (**C**all**e**e-only **i**nference and **ve**rification), a practical and effective solution that leverages callee-only capability to defend against caller ID spoofing. CEIVE explores the simple solution concept of initiating a callback session to the originating phone number and comparing the call states of the outgoing call session with the incoming call. The goal is to verify whether the claimed caller ID indeed matches the actually used one. Specifically, upon receiving an incoming call request session *inCall* with a potentially spoofed phone number, the (victim) callee initiates a callback session *auCall* before accepting the incoming call. In the absence of ID spoofing, the callee of *auCall* is identical to the caller of *inCall*[1], the user then accepts the call. In the presence of ID spoofing, *auCall* will reach another party different from *inCall.caller*, and the user can consequently reject the call. To make the verification more reliable, we have to infer the fine-grained call state of *auCall.callee*, (e.g., *dialing, idle, on-a-call (connected)*), in order to assert whether it matches the anticipated one of *inCall.caller*. This motivates us to exploit an available, yet unexplored side channel. Another salient feature of CEIVE is that, it has to infer the *inCall.caller* state based on the victim-side information only. This is because the caller can be malicious.

CEIVE formulates the core design as an inference problem, and further devises novel techniques to address three practical challenges. First, inferring the call state of auCall.callee *only from auCall.caller's observation* is difficult. We find that common features (e.g., call/phone states) from mobile phones would not work; They can inform the *local* caller's state, but are insufficient to infer the *remote* callee's call state. To this end, we discover an unexplored side channel of call setup signaling, and show that it is feasible to infer certain call state out of the sequence of observed signaling messages (§3). Second, inference accuracy is affected by *many factors unknown to auCall.caller*. We find that the sequence of call setup signaling messages varies with carriers, call technologies, call settings, and even seemingly-random factors (controlled by network operations). For example, the same sequence is observed for two distinct call states (e.g., dialing or being-dialed); Multiple sequence variants are observed for the same call state. We thus enhance spoofing verification with inference, and design an inference engine tailored to caller ID spoofing detection. In doing so, we enable a coarse-grained inference to learn a few, but not all call states, and show that they suffice to differentiate `spoof` from `no-spoof` in most usage scenarios (§4.2 and §4.3). Third, single inference-verification may still fail to resolve ambiguity in certain scenarios, especially when the adversary designates special attacks against CEIVE to manipulate call states (e.g., making the authentic caller busy or stuck at certain hard-to-differentiate state). CEIVE thus employs multiple-phase (mostly two) verification, and leverages delta and coherence across phases to refine inference (§4.3). Finally, it applies re-learning for automated evolution (§4.4).

We have prototyped CEIVE on Android phones, and validated it with real-world evaluations. We first show that CEIVE successfully combats caller ID spoofing used in a recent real scam call. We further run controlled experiments to launch a variety of caller ID spoofing attacks (C2-C7), as well as normal calls (C1). We test with both 4G voice solutions: VoLTE and CSFB (described in §2). Surprisingly, CEIVE is 100% accurate in almost all attack scenarios (except C7) with all top four US mobile and one landline carriers. In the worst case (C7: a targeted attack against CEIVE), CEIVE just fails without deterministic inference and reaches an ambiguity outcome (N/A) under certain settings, which still can keep the victim stay alert. Moreover, CEIVE is fairly responsive and completes within tens of seconds (up to 23 seconds in our tests). It defers call answering only for several seconds (mostly within 4-10 seconds for VoLTE or 8-10 seconds for CSFB). It is user friendly without degrading normal call experiences. It is extensible to new carriers with learning ability.

We have also noted that the current design of CEIVE does have a limitation. It does not work with a multi-line caller, where multiple phone lines share the same number; this is part of our future work. Nevertheless, CEIVE offers the first callee-based solution against caller ID spoofing. It does not require additional infrastructure support or cooperation from the caller side (who can be malicious). Mobile users, who are potential victim callees, have strong incentive to deploy the solution. While CEIVE is currently targeting 4G phone users, it is conceptually applicable to 5G/3G/2G mobile networks and even non-cellular calls.

[1]It may not be true in multi-line telephone systems, which are discussed in §7. In this paper, we consider a single-line system where one phone number matches one entity only.

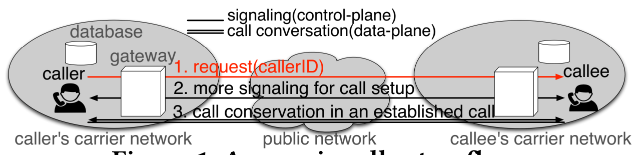
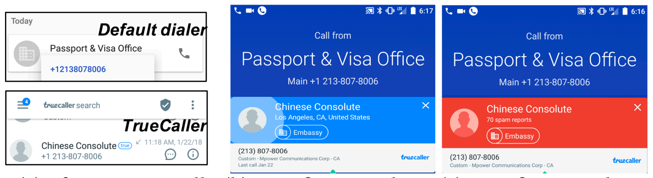


Figure 1: A generic call setup flow.



(a) after a scam call (b) spoofing 20D later (c) spoofing 20D later

Figure 2: Screenshots of the callee phones (Pixel 2) in a real scam and two controlled caller ID spoofing tests.

2 CALLER ID SPOOFING ATTACK: EASY TO LAUNCH, HARD TO DEFEND

We confirm that caller ID spoofing is indeed easy to launch but hard to defend over 4G LTE networks.

Background. Figure 1 depicts a generic call setup flow for any call technology. Call signaling runs first to establish a call session and then starts voice conversations over the session. The signaling starts with a setup request from the caller to the callee, followed by more signaling required by the call setup procedure. Both parties obtain call service from their own carrier network (CN). CNs are inter-connected so that call parties from different CNs can talk to each other.

We consider the callee's CN as the 4G LTE network. 4G supports two voice solutions: Voice-over-LTE (VoLTE) [4] and Circuit Switched FallBack (CSFB) [13]. VoLTE adopts Voice-over-IP (VoIP) and carries voice calls (and its signaling) in IP packets; CSFB leverages legacy 3G/2G networks to provide CS voice calls. Both conceptually support similar signaling but use different protocols. VoLTE uses Session Initiation Protocol (SIP) [43], while CSFB uses Call Control (CC) [12][2]. Though they speak different protocol languages, e.g., the first request via `INVITE` in SIP for VoLTE and via `SETUP` in CC for CSFB/CS, the translation is handled by their border gateways for inter-operability. For example, `INVITE` is mapped into `SETUP` once leaving 4G and entering 3G/2G. More signaling messages will be introduced in §3.

Each call party has a globally unique ID, often a telephone number (e.g., +1 xxx-xxx-xxxx)[3]. ID acts as a permanent address-of-record which is assigned upon subscription and is authenticated before use. Specifically, cellular networks run Authentication and Key Agreement (AKA), which uses the shared secret key stored at SIM (locally) and known only to the operator (user database) to authenticate each other.

Caller ID Tests: Easy to spoof. Caller ID spoofing uses a fake ID. In this paper, we consider the spoofing scenario where caller Eve (E, hereafter) calls the victim Bob (B) by fabricating Alice's ID (A.ID).

In reality, caller ID spoofing is technically feasible and simple. Spoofer E simply alters the caller ID carried in the setup request, which is allowed through because E's CN does not enforce the forwarded caller ID is the same as the authenticated one. Use VoIP/VoLTE as an example. It uses the '`From`' header in the `INVITE` message to convey the caller ID, as illustrated in Figure 4d. E places A.ID instead of E.ID, so that B only sees an incoming call from 'A'. Even worse, caller ID spoofing is even offered as one public service by fake ID providers such as Spoofcard [47], Spooftel [48], FakeCall [10] and many alike apps. To use it, E only needs to input B's phone number as the target one and A's phone number as the desired spoofed caller ID. Spoofing takes a few seconds and zero cents; it is very easy to use.

Figure 2 shows that users can indeed easily launch a spoofing attack. We use Spoofcard [47] and FakeCall [10] and successfully make spoofing attacks towards our test phones (Pixel 2) in Figures 2b and 2c. The same ID was used in a recent real scam call, which has resulted in more than $1M loss for a single victim [8, 32, 41, 55]. We also observe that the solution to combating caller name spoofing, e.g., TrueCaller [51] and Google's dialer [5], might not work. Both failed to detect when the real scam call happened in Jan 2018 (Figure 2a), though it was reported 6 months ago [41, 55]. In our spoofing tests 20 days later, TrueCaller worked to certain degree for the phone with Internet access (Figure 2c), but failed at the phone without Internet access (Figure 2b). Google's dialer failed on both phones but worked at another Pixel 2 (omitted due to space limit); it is unclear why it fails. These solutions make things even worse, because they likely mislead the callee to trust the call is from an authentic party.

In our controlled experiments (both attackers and victims owned by us), we test with fabricating other phone numbers (mobile or landline, personal or business, from different states and countries, >100 in total). We confirm that, all are easy to spoof with no sign of restrictions.

Lessons and root causes. We discover that, spoofing is feasible even when user authentication is in place. Although authentication is a well-known technique against spoofing, two reasons make it fail to prevent caller ID spoofing. *First, user authentication is within the caller's network, but not end-to-end.* There are no means for the callee to authenticate the caller; As long as ID spoofing is permitted in the caller's CN, the callee has to 'trust' the received ID. *Second, user authentication is separated from call setup signaling.* Although authentication runs at the start (to authorize call making), no mechanism prevents the caller from altering the forwarded ID, thus hiding its authenticated ID, during later call setup.

[2]UMTS/GSM uses CC and CDMA uses similar signaling [28].

[3]Other IDs like username (e.g., xxx@xxx.xxx) are allowed in VoIP. In this paper, we consider telephone numbers only.

Related work: Hard to defend in reality. There are several solution proposals without actual deployment. They address the issue of end-to-end authentication via a global authority (e.g., a public certification service [22, 24, 27, 35]) or a public key infrastructure (PKI) [52], to authenticate each party before call setup. [46] addresses the second issue through additional network assistance (authentication required at gateways during call setup). They are deemed effective in principle but have not been deployed. Its real use is not foreseen in the near future, due to its deployment costs (third-party global infrastructure, network infrastructure upgrade, changes on every phone). An alternative solution is to detect caller ID spoofing [38, 39]. These proposals use challenge-and-response between two ends and require the caller to respond to SMS [39] or a call [38]. They require the caller's cooperations, which may mandate updates on all phones (i.e., all possible callers); Moreover, they only consider simplistic attacks where E does nothing to A.

3 BASIC IDEA AND FEASIBILITY STUDY

We now present the basic idea of Ceive, and conduct feasibility tests while identifying practical issues. We consider VoLTE first and defer CSFB to §3.5.

Threat model. We consider a large class of practical spoofing attacks against mobile phone users. The attack is initiated by malicious users, who have full control of their phone devices. It is not from mobile carriers. The attack can be launched by leveraging public service/software or running private programs for designated attack operations. The adversary can not only make a spoofed call request to the victim callee, but also manipulate other call parties through legitimate access interfaces for advanced attacks. For example, the attacker can dial the true caller (here, A) or even establish another call with A accompanying the spoofed call; The adversary can further adjust attack frequency and modify dial/call operations. However, the attacker has no ability to hijack or compromise the victim's phone, the true caller's device, or their carrier networks. No malware can be installed; The true caller does not conspire with the adversary; The carrier network infrastructure also functions well. In short, the victim callee, the true caller and their carrier networks are all trustworthy.

3.1 Basic Idea

Our basic idea (shown in Figure 3) is to verify whether the caller ID (A.ID) is spoofed or not, by comparing the call states of two call sessions. For an incoming call *inCall*, Ceive asks the callee (i.e., B) to make an *auCall* back to the originating ID (①). B makes use of the *inCall*'s context to infer the state of caller X (A or E in the absence/presence of spoofing). For example, X is dialing when *inCall* rings. Meanwhile, B uses

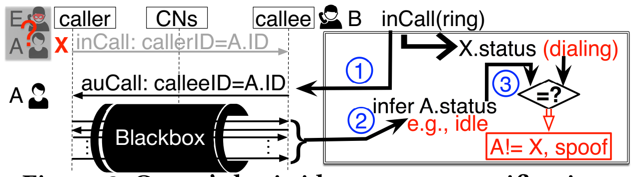


Figure 3: Ceive's basic idea: one-run verification.

its own observation on *auCall* to infer A's call state (②), and compares it with X's (③). If they mismatch, A is asserted to be not X, and spoofing happens to *inCall*.

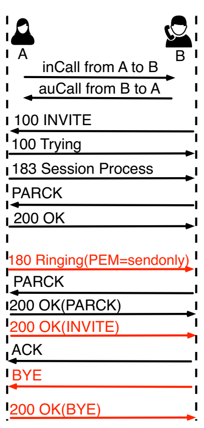
The above simple solution concept has several nice features. No control is assumed on other components (the carrier infrastructure, or other devices). It does not require cooperation by others or extra information access. It also works under two premises: (1) B's observation is able to infer A's distinct call state. When the call state of *auCall.Callee* changes, the observation at *auCall.Caller* should also change to make the inference possible. (2) The inferred A's call state should differ from the true call state *at least once* upon spoofing.

We next conduct feasibility tests to address a key technical issue: *what available information from the auCall.caller side can be used to infer the distinct state on the remote auCall.callee side?* We first study common call information provided by mobile OSes (using Android as an example) such as PRECISE_CALL_STATE[18], PHONE_STATE in TelephonyManager [19] and system logs. However, we conclude that such information fails to infer the state on the *remote* callee side, because it only provides call states on its *own* side. In principle, both sides are in the *same* call session, and the caller should be able to know what happens at the terminating party. However, in practice, these high-level APIs hide internal, fine-grained call context and fail to run inference required by Ceive. We thus look into raw call context information. We find that, the sequence of call setup signaling messages (SIP for VoLTE and CC for CSFB/CS) suffices!

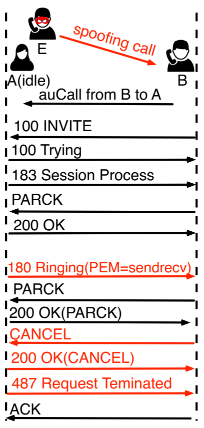
3.2 Baseline Feasibility Tests

We first run basic feasibility experiments to validate that, the call signaling messages received on the caller's side are enough to infer the callee's call state. We run our experiments in three common call settings:
(C1) A calls B (no-spoof),
(C2) E calls B while A is idle (spoof-idle),
(C3) E calls B while A is on a call (spoof-conn).

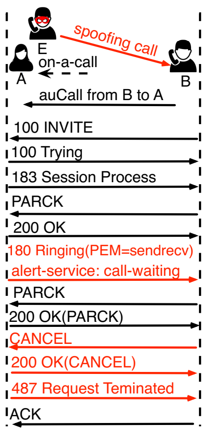
We collect SIP signaling messages for *auCall* using tcpdump at phone B, a rooted Android device. We have tried with 10+ phone models (from Samsung, Google, LG, Motorola, Xiaomi, etc) and found no difference. We also test with all four top-tier US carriers: AT&T, T-Mobile, Verizon and Sprint. The observations are slightly different, but all are proven feasible (Figure 5, in §3.4). We use the T-Mobile results to illustrate Ceive's feasibility.



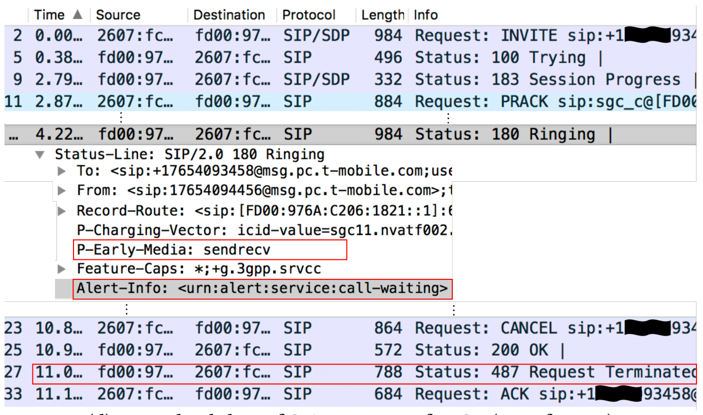
(a) C1: no-spoof



(b) C2: spoof-idle



(c) C3: spoof-conn



```
    Time ▲   Source     Destination  Protocol  Length  Info
 2  0.00…    2607:fc…   fd00:97…     SIP/SDP     984   Request: INVITE sip:+1[redacted]934
 5  0.38…    fd00:97…   2607:fc…     SIP         496   Status: 100 Trying |
 9  2.79…    fd00:97…   2607:fc…     SIP/SDP     332   Status: 183 Session Progress |
11  2.87…    2607:fc…   fd00:97…     SIP         884   Request: PRACK sip:sgc_c@[FD00
                 ⋮                                          ⋮
 …  4.22…    fd00:97…   2607:fc…     SIP         984   Status: 180 Ringing |
  ▼ Status-Line: SIP/2.0 180 Ringing
    ▶ To: <sip:+17654093458@msg.pc.t-mobile.com;use
    ▶ From: <sip:17654094456@msg.pc.t-mobile.com>;t
    ▶ Record-Route: <sip:[FD00:976A:C206:1821::1]:6
      P-Charging-Vector: icid-value=sgc11.nvatf002.
      P-Early-Media: sendrecv
    ▶ Feature-Caps: *;+g.3gpp.srvcc
      Alert-Info: <urn:alert:service:call-waiting>
                 ⋮                                          ⋮
23  10.8…    2607:fc…   fd00:97…     SIP         864   Request: CANCEL sip:+1[redacted]934
25  10.9…    fd00:97…   2607:fc…     SIP         572   Status: 200 OK |
27  11.0…    fd00:97…   2607:fc…     SIP         788   Status: 487 Request Terminated
33  11.1…    2607:fc…   fd00:97…     SIP         684   Request: ACK sip:+1[redacted]93458@
```

(d) Wireshark log of SIP messages for C3 (spoof-conn)

Figure 4: Examples of SIP message sequence (key in red) in three call scenarios via VoLTE in T-Mobile.

	True state	Key observations (Features)
C1	A is dialing	*180.PEM* = sendonly
C2	A is idle	*180.PEM* = sendrecv
C3	A is conn	*180.PEM* = sendrecv, *180.ALERT* = call-waiting

Table 1: Examples of key observations in T-Mobile.

Figure 4 plots the diagrams of SIP signaling messages observed at B in C1-C3 scenarios. *auCall* is initiated, when *inCall* rings but is not accepted by B. We make three observations. First, the sequences of call signaling messages share many common parts in all three scenarios. Specifically, all start with `INVITE`, followed by $100 \rightarrow 183 \rightarrow \cdots \rightarrow 180 \cdots \rightarrow 200 \cdots$. These numbers represent the SIP state and response codes, all of which are standardized [43]. Second, each sequence contains certain critical information to distinguish three call settings. For example, in the received `180 Ringing` message, there are two fields: `P-Early-Media` (PEM) and `Alert-Info` (detailed logs in Figure 4d). Table 1 lists their distinct values in all three scenarios. Third, we also discover redundant features which can infer distinct call state as well. C1 observes `200` but C2/C3 uses `487 Request Terminated` in response to `INVITE`; C1 uses `BYE` while C2/C3 uses `CANCEL` at the end.

We thus exploit the unexplored side channel of call setup signaling messages. We consequently infer distinct callee state: `dialing` (C1), `idle` (C2) and `conn` (C3) (Premise 1) while the inferred state in C2/C3 differs from the anticipated state in the absence of spoofing (C1) (Premise 2).

3.3 Why Should it Work?

We now explain why the above solution should work. The rationale lies in the call setup procedure standardized by Internet RFCs and cellular specifications. Table 2 exemplifies such important information.

First, call setup signaling messages contain explicit or implicit information related to the callee's state to facilitate the call setup. The call request can reach the callee if (s)he is available. When the callee is busy, we can learn it from the ringtone or being switched to the voice mailbox. If the callee has call waiting, the call request can still reach him/her if (s)he is in a call.

Second, standard specifications mandate a rich set of signaling messages, which carry rich context information and can be exploited to infer the call state on the other party. SIP defines many parameters and response codes [31, 43] (125 parameters and 50 codes). For instance, '`180 Ringing`' indicates that the call request arrives at the callee; '`181 Call Is Being Forwarded`' is used when the call is forwarded to a voice mailbox for a busy callee;'`486 Busy Here`' indicates a busy callee. Moreover, SIP defines extensions to convey more information. For example, the `P-Early-Media` (PEM) field [1] authorizes early media (e.g., ringtone), with '`sendrecv`' indicating a bidirectional line, '`sendonly`', '`recvonly`' and '`inactive`' indicating a directional line to the caller, from the caller, and no line. Another example is `URN-Alert` (Figure 4d), which provides common understandings of the referenced tones [3]. '`call-waiting`' indicates that the callee is in an active or held call, and '`forward`' indicates the call will be forwarded.

Third, these signaling messages are associated with the call setup's finite state machine (FSM), which together reveals more call state information. For instance, '`487 Request Terminated`' implies that the request was terminated by a `BYE` or `CANCEL` request [43]. The caller sends `CANCEL` when planning to terminate a call before the call is answered. It sends `BYE` if the original `INVITE` still returns '`200 OK`'. `CANCEL` is observed without '`200 OK`' in response to `INVITE`. Cellular specifications [15–17] also assert that VoLTE adopts certain signaling messages useful for callee state inference from the caller-side observations.

In summary, call setup uses a stateful FSM and its signaling likely carries enough information to infer callee state. This

Field	Reference: Values (examples)
SIP response codes	RFC3261[43]: e.g., 200 OK, 180 Ringing, 181 Call Is Being Forwarded, 182 Queued, 183 Session Progress, 301 Moved Permanently, 480 Temporarily Unavailable, 481 Call/Transaction Does Not Exist, 486 Busy Here, 487 Request Terminated, · · ·
PEM	RFC5009 [1]: sendrecv, sendonly, recvonly, inactive
URN-Alert	RFC7462 [3]: normal (default), call-waiting, forward, recall:callback, recall:hold, recall:transfer, · · ·
VoLTE FSM	TS24.229[15], TS24.628 [17], TS24.615[16]: e.g., carrying early-media value or alert-info in 180/183, call terminated by network when busy· · ·

Table 2: Main standards on VoLTE (VoIP) call setup.

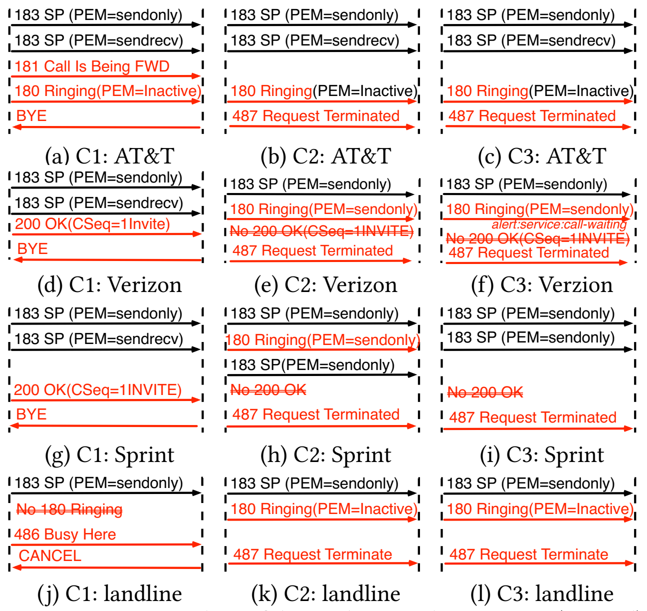


Figure 5: Examples of key observed patterns (in red) when A is from other three US carriers and landline.

makes callee state inference possible based on the signaling message sequence observed on the caller side.

3.4 More Feasibility Tests

We run more real-world tests to see whether the idea works in more usage settings. Figure 5 shows the selected key patterns when B remains the same but A is from other three US carriers and one landline. AT&T and Sprint do not support VoLTE but use CSFB/CS. We test with many other settings (see §6) The feasibility is validated in all settings. We summarize four findings and discuss their design implications.

First, our idea works in all tested carriers, despite with distinct key patterns (in red). For instance, in AT&T, C1 has `181` prior to `180`, which can differentiate itself from C2 and C3. In Verizon, C1 has no `18x` response code but `200 OK` in response to `INVITE` followed by `BYE`. This implies carrier-specific implementation. The standard stipulates the mechanism, but leaves options for vendors and carriers. The carrier-specific diversity makes inference more involved, as B may not know A's carrier information (at least initially).

Second, we observe redundancy when recognizing distinct call states. This offers more design choices and makes our inference more robust (elaborated in §4.2). For example, to differentiate C1 and C2, AT&T can use either `181`→`180` versus `180`, or `BYE` versus `487`, or both. Verizon may rely on `180`, or `BYE` versus `487`, or both. We further observe another level of redundancy in the message sequence. For example, `CANCEL` and `487` are associated (see Figure 4), because `487` is invoked by `CANCEL` in FSM. We can use `487` only, `CANCEL` only, or both, for our inference. It also helps us to infer A's carrier information.

Third, some call states are not recognizable in certain scenarios. In AT&T, the observed sequences are the same when A is idle (C2) or connected (C3). The verification still works when *auCall* is made when *inCall* just rings at B. However, it may fail when *auCall* is made after B accepts the call (X is conn). The same message sequence is observed in all three (`spoof` and `no-spoof`) settings and we cannot infer A to be `conn` or `idle`. The adversary may consequently upgrade his spoofing strategy. For example, E uses another channel to dial A and affect A's state to defeat B's verification. This calls for an effective solution over coarse-grained state inference to handle rich possibilities (elaborated in §4.3).

Last, our idea also works when the callee (A) is a landline phone. C1 has distinct patterns from C2 and C3. B receives a response `486 Busy Here`. In our tests, the response codes may change (e.g., `481`), but C1 is consistently observed without `180` or `487`, both of which appear in C2 and C3. Landline is known to use different signaling. The signaling translation between A and B is thus inevitable. Our study shows that critical information on signaling is still retained after the translation. Call technologies follow similar setup conventions. This makes our approach quite promising in practice.

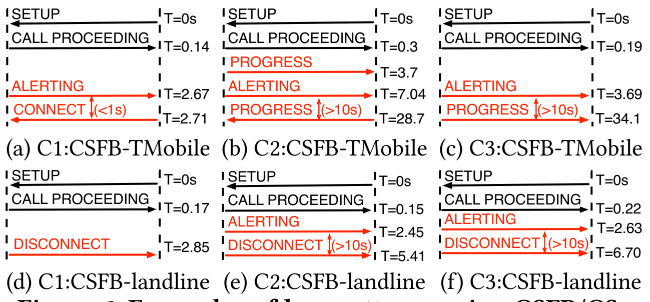


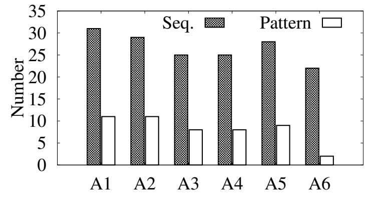
Figure 6: Examples of key patterns using CSFB/CS.

3.5 Feasibility Tests on CSFB/CS

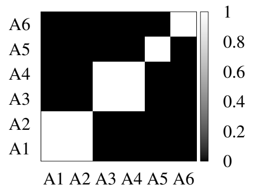
Our idea is also applicable to CSFB/CS calls. Figure 6 shows the key patterns with CSFB, where B uses AT&T and A

A1	Dialing
A2	Being dialed
A3	Connected (not-on-hold)
A4	Connected-on-hold
A5	Idle
A6	Unreachable (off, flight mode, invalid number, etc)

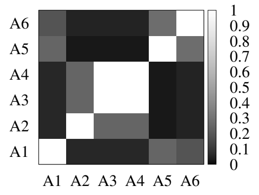
(a) Tested callee state



(b) Number of sequence variants



(c) Similarity (T-Mobile only)



(d) Similarity (all 4 US carriers)

Figure 7: Challenges for CEIVE.

uses T-Mobile or landline. We do not show results for other combinations due to space limit, but all are feasible. This matches our expectation, because CSFB follows the call setup convention and conveys state information in the signaling messages. We also run experiments directly with CS calls over 3G and observe no difference between CSFB and CS calls. To retrieve CSFB/CS call setup signaling messages, we use MobileInsight, an open-source tool [36]. We plot important call control (CC) messages, such as `SETUP`, `ALERTING` and `CONNECT`, with more regulated in [14].

We have four findings. First, compared to VoLTE, CSFB/CS carries less information in its signaling without second-level messages such as '`P-Early-Media`' and '`URN-Alert`'. Second, it still carries sufficient information to distinguish certain call states. In the landline case, C1 has no `ALERTING`, which differs from C2 and C3. In T-Mobile, C1, C2 and C3 observe different signaling sequences: `ALERTING-CONNECT`, `PROGRESS-ALERTING-PROGRESS`, and `ALERTING-PROGRESS`, respectively. Third, we also observe redundant features in CSFB. In these examples, time interval information is not necessary. In the T-Mobile case, the `ALERTING-CONNECT` interval in C1 is small ($<$ 1 second), and the interval between `ALERTING` and the second `PROGRESS` in C2 is always larger than 10 seconds. Figure 6 plots time information for one run, but the interval patterns have been confirmed in all runs. Last, we detect less carrier-specific diversity. Cellular carriers follow similar signaling procedures in CSFB/CS, as confirmed by the standards [14]. In summary, inference is still feasible in CSFB/CS calls despite smaller signaling space than VoLTE.

3.6 Remaining Issues

Our study also uncovers three practical issues to be addressed. First, the observed signaling sequence may vary for a given callee's state. For example, when A uses CSFB but not VoLTE, a second `183`, rather than `180`, is observed. The observed sequences are affected by many factors unknown to the caller. They include the callee's carrier, callee's VoLTE/CSFB technology, voice forward configuration, additional call service etc. We further test with six typical callee states A1-A6 (dialing, being dialed, connected-not-on-hold, connected-on-hold, idle, unavailable, see Figure 7a). Figure 7b shows the number of unique sequences observed (the number of pattens greatly reduces thanks to our feature extraction in §4.2). Moreover, the same sequence exhibits for distinct call states. We use Jaccard index to quantity similarity (dissimilarity) when inferring distinct call states. Let S_{A_i} denote the set of sequences for label A_i, and $J(S_{A_i}, S_{A_j}) = \frac{|S_{A_i} \cap S_{A_j}|}{|S_{A_i} \cup S_{A_j}|}$. Figures 7c and 7d plot the similarity matrix using T-Mobile only and all four US carriers. If the same sequence is observed under two callee states, they are not differentiable. We find that, 'dialing' (A1) and 'being dialed' (A2) are not distinguishable; 'connected-not-on-hold' (A3) and 'connected-on-hold' (A4) have almost the same patterns. Note that, it is not necessary to distinguish all call states (elaborated in §4.2). Our solution is 100% reliable for a single carrier, but accuracy might slightly reduce (still very high) once A is from another carrier unknown to B. For instance, when A is idle and connected in AT&T, there is no difference in the observed sequences.

Second, user operations may incur uncertainty as well. We infer the call state at the start (ring$\longmapsto$dialing), but it may change during verification (e.g., A accepts `auCall`). Consequently, the captured sequence might not be for a single state. We do observe different sequences when A accepts, rejects, or does nothing to `auCall`. For example, the sequence ends with `200` (`INVITE`) and `200` (`BYE`) when A accepts the call. Despite such dynamic factors, it is still feasible to infer the callee state, because we can infer A's response from the observed sequence and use this hint in state inference.

Third, the adversary may exploit more sophisticated spoofing strategies. For example, it may dial A while dialing B, thus manipulating A's state and deceiving our verification. It is desirable for CEIVE to withstand such advanced attacks.

4 CEIVE DESIGN

We now present the full design of CEIVE, which extends the above feasibility study to address practical issues. CEIVE seeks to exploit the callee-side capability only to timely verify whether an incoming call is truly associated with its authentic caller ID. Central to CEIVE is to let the callee to *proactively* and *strategically* (elaborated later) to call back (*auCall*) to the originating ID until the spoofing hypothesis

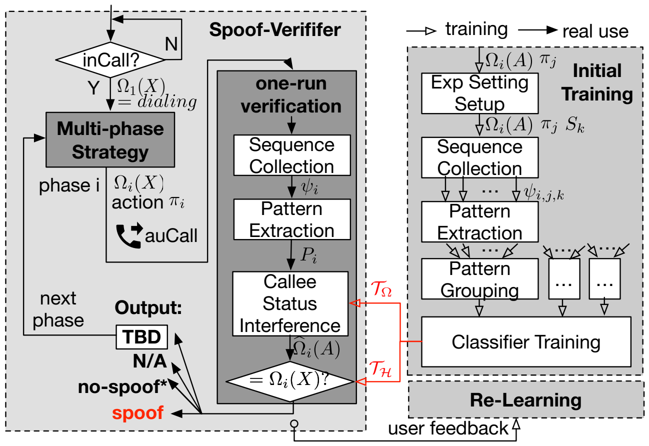


Figure 8: Overview and operation flows of CEIVE.

$\mathcal{H}$ is validated.

$$\mathcal{H} : \begin{cases} True, & inCall.Caller \neq auCall.Callee \\ False, & inCall.Caller = auCall.Callee \end{cases} \quad (1)$$

Once $\mathcal{H}$ is accepted, *inCall* is marked as spoof; otherwise, no-spoof. The original problem is to validate whether *inCall.CallerID* is associated with *inCall.Caller*. Because each caller ID matches only one unique entity, *auCall* reaches the callee associated with *inCall.callerID* as long as the carrier functions normally. In order to validate $\mathcal{H}$ and assert *inCall* is a *spoof*, we only need to observe one mismatch between their call state captured, Namely,

$$\exists i, \Omega_i(inCall.Caller) \neq \Omega_i(auCall.Callee) \longmapsto \texttt{spoof}, \quad (2)$$

where $\Omega_i(X)$ denotes X's call state at time i. Otherwise, we believe it as no-spoof when they match every time;

$$\forall i, \Omega_i(X) = \widehat{\Omega}_i(A) \longmapsto \texttt{no-spoof}. \quad (3)$$

We apply rules (2) and (3) to infer spoofing in CEIVE.

4.1 Overview of CEIVE

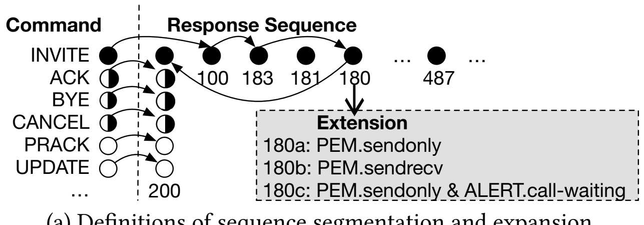
Figure 8 illustrates the overall design and main operation flows of CEIVE. The core module is *spoof-verifier* for runtime spoofing detection when a call comes. We devise a multi-phase (mostly two-phase) verification strategy because one-run verification described in §3.1 may not always suffice in practice. The initial phase starts with $\Omega_1(X) = dialing$, namely, we dial the first *auCall* while *inCall* is ringing; At each phase possibly with distinct $\Omega_i(X)$ (dialing or connected), we perform one-run verification as illustrated in §3.1. Specifically, we perform an action π_i (make one *auCall*) and exploit the received sequence of call setup signaling messages to infer $\widehat{\Omega}_i(A)$, the state of the originating ID and determine whether the spoofing incurs by comparing with $\Omega_i(X)$. We apply Eqn. (2) to ascertain spoof. Otherwise, when both states match at all the phases, we believe it as no-spoof*. We cannot be 100% confident in no-spoof inference because CEIVE uses a limited number of phases; This cannot guarantee Eqn. (3) holds true in any case. Due to the attacker's manipulation and the existing uncertainties, a match is possible when $X \neq A$ (spoof). The details are elaborated later. Consequently, it is to be determined (TBD) when a match is observed and not all the phases complete. If so, the next phase will be invoked accordingly, for example, making another *auCall* when *inCall* is answered and $\Omega_i(X) = conn$.

The modules of *initial training* and *re-learning* are to train and update decision tree rules (classifiers) used by *spoof-verifier*. The former is mandatory and requires one-time effort before use. The latter is optional and can update rules with user feedbacks (labelled samples) after use. We need to learn these rules for two reasons. First, our raw observation is a sequence of messages which has a relatively high dimension, while the effective patterns lie in a much smaller subspace. The rules based on the original sequence is prone to more variants (caused by irrelevant messages) and this becomes harder to bootstrap accurate classifiers ($\mathcal{T}_\Omega$ and $\mathcal{T}_H$) for call state inference and $\mathcal{H}$-validation, especially using a small number of samples at the start. To this end, we seek to leverage domain knowledge on call setup and extract useful features (sub-sequence) for CEIVE's need. Second, no single rule can fit all. We must handle sequence variants in reality and ensure its effectiveness under a variety of unknown factors like caller's carrier, call technology, call configurations and operations etc. We find that the rules are specific to the victim callee's carrier and call technology (VoLTE or CSFB/CS). There is no surprise that the rules learned can be shared to the same-type mobile users using the same carrier and call technology. The training effort is modest. We next elaborate on each component.

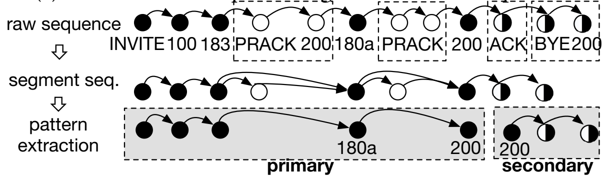
4.2 Initial Training

It takes three steps: sample collection, pattern extraction, and classifier training.

Sample collection. We design and conduct experiments to collect samples to infer the callee status from the caller's observation. We find that inference is not affected by B's *inCall* status (whether it is on another ongoing call or just idle), so we consider the *auCall* call session only. Given the output label of the callee's status $\Omega_i(A)$, we collect training samples under two settings: the caller's call action π_j under control and typical experimental settings S_k which might be unknown in use. We consider four output labels: dialing (A1), connected (A3), idle (A5) and unavailable (A6) (Table 7a). This is because our preliminary study shows that A1 and A2 are not distinguishable, while A3 and A4 are almost indistinguishable in reality. Action π_j is constrained by the caller's power and we consider VoLTE and CSFB/CS



(a) Definitions of sequence segmentation and expansion



(b) Using the example of Figure 4a (T-Mobile, VoLTE, C1)

Figure 9: Sequence pattern extraction.

calls. Experimental setting S_k takes into account other factors such as the callee's carrier and call technology (VoLTE, CSFB, landline), voice service configuration, etc. In each run r, we collect one raw sequence sample for $\psi_{i,j,k}[r]$.

We later find that there is no need to enumerate all possible experimental settings (which is extremely hard, if not impossible). In fact, our training quickly converges with several samples in typical settings. This is because the key patterns are commonly observed due to the inherent FSM.

Pattern extraction. We next extract low-dimensional features out of the raw sequences for further inference. We take a domain-specific approach over two facts: (i) the sequence of signaling messages is structural (determined by its inherent FSM); (ii) many segments are common, but only a few distinct segments are critical to inference.

Our extraction has two steps. First, we represent the raw sequence into a simple and meaningful manner. Figure 9a illustrates the segment structure. Each segment starts with a signaling command or request (e.g., `INVITE`, `ACK`, `OPTION`, `BYE`, `CANCEL`,`PRACK` and `UPDATE` [43]), and ends with its response codes (zero, one or multiple). As a result, each segment has its call signaling context. The `INVITE` segment is used to invoke call signaling, while the `ACK/BYE/CANCLE` is to stop signaling. Other segments like `PRACK` and `UPDATE` are used for other purposes and irrelevant to call status inference (○). We also find that, all segments except `INVITE` have at most one response code (usually `200` or no response). This implies that only its segment head (aka, the request itself) suffices. The `INVITE` request not only invokes multiple response codes, but also exhibit complicated patterns, such as `183-183-487`, `183-180-487`, `183-180-200`, and so on. Moreover, certain response codes have multiple variants such as `180/183` (with distinct `PEM` and `ALERT` values). They thus obtain multiple extensions based on the additional information carried. Figure 9b illustrates how it works using the T-Mobile example (Figure 4a). We represent the raw sequence (top)

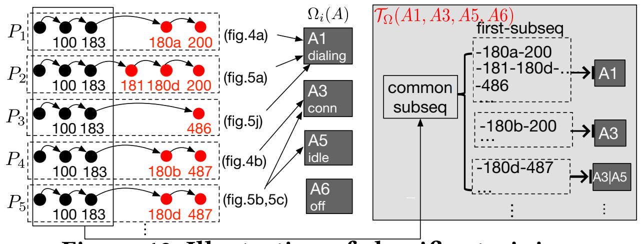


Figure 10: Illustration of classifier training.

into a segment sequence by substituting all non-`INVITE` segments with its command head only.

Second, we extract the pattern in the form of one primary segment (`INVITE`), along with one secondary segment. Primary and secondary segments are defined based on their importance to inference. It is not surprising that the `INVITE` segment plays an essential role (●). Other segments like `ACK/BYE/CANCEL` are somehow useful and act as the secondary ones (◑). Note their significance is not only justified by their meanings, but also is confirmed in the training process. Finally, we retrieve the pattern as one sole `INVITE` segment (here, `INVITE-100-183-180a-200`) and a chain of the secondary ones (here, `ACK-BYE`) followed by a primary segment element, here `200`(for INVITE), which records how to chain two segments. This way, we greatly reduces the feature space while still retaining key information.

Classifier training. The last step is to train the classifiers $\mathcal{T}_\Omega$ and $\mathcal{T}_H$. Their approaches are similar and the only difference is that the latter is a binary classification, which is simpler. Given $\Omega_i(A)$, we only consider two sets: match ($\Omega_i(A)$) and dismatch ($\neg\Omega_i(A)$). In the former $\mathcal{T}_\Omega$ training, we handle multiple labels (here, A1, A3 and A5 and A6). We use the former task to present our training procedure. Consider the classifier is call technology specific. The training runs separately per π_j (VoLTE or CSFB/CS).

Figure 10 illustrates the training procedure using real instances described in Figures 4 and 5. The training input is a bipartite graph which maps the pattern to the status label. Note that not every pattern corresponds to single label. For example, pattern P_5 is observed in both A3 (conn) and A5 (idle). This results in ambiguity, so we cannot precisely tell the callee status in use. So our training is to remap all the patterns to new status labels so that every pattern contains no ambiguity. This is crucial for the subsequent spoofing inference. CEIVE utilizes it to determine the confidence level of our inference (described later). To do so, we first group all the patterns per $\Omega_i(A)$ and then divide these groups into exclusive sets. For those patterns which are associated with multiple labels, we create a new label. For example, P_5 is labelled as A3|A5, which is different from A3 only or A5 only. Initially, these exclusive sets can be obtained via set interaction and difference. For two sets A and B, we divide into three new sets: $A \cap B$, A/B (A only), B/A (B only). Theoretically, we convert n (here, n=4) groups into at most $2^n - 1$ ($= C_n^1 + C_n^2 + \cdots + C_n^n$) sets. It can work interatively. When a new sample (P_x, A_x) comes, we first check if the extracted pattern is new. If yes, P_x will be added into its A_x-only set. If no, we check if its new label conflicts with the existing label (say A_{old}. When the new label is not included, we need to move this patten into the set labelled as $A_x|A_{old}$. In fact, we iteratively perform the above process until we finish all the training samples.

To make Ceive efficient, we take two measures in training. First, we locate *common subsequences* that appear in all patterns for distinct callee status. They are of no value for inference. We thus apply the popular LCS (Longest Common Subsequence) algorithm [33]. We run it iteratively until we find all common subsequences. In this example, we identify a common subsequence of the first three messages (`INVITE-100-183`). Second, our classifiers use the first distinct subsequence, rather than the whole pattern sequence. We apply FreeSpan, a sequential pattern mining algorithm [30] to generate unique subsequence patterns. Note that the first distinct subsequence is sufficient to classify different callee statuses. For example, after the common subsequence, `180a` or `181` or `486` infers A1 (dialing) but `180d` indicates A3|A5. This speeds up spoofing interference without waiting for all the messages. We notice that the first distinct subsequence may vary as training samples grow. One pattern may change its label upon a new sample. To handle this, we still perform the training for the whole pattern sequence (primary and secondary segments). Once the first subsequence expires, we leverage the rest subsequences (redundant features) to update the first unique subsequence.

4.3 Spoof Verifier

The module of *Spoof Verifier* has two main components: multi-phase verification strategy and one-run verification.

One-run verification. Each verification starts with known $\Omega_i(X)$ and actions π_i at phase i. Following the flow of Figure 8, it uses many common components in the training process. We focus on three distinct operations.

First, we go directly for spoofing inference. We check whether the observed pattern matches $\Omega_i(X)$, without inferring $\hat{\Omega}_i(A)$. Moreover, $\Omega_i(X)$ may not be an arbitrary state of Table 7a. Due to the incoming call constraints, there are only two (actually three) options: (1) when *inCall* still rings, $\Omega_i(X)$ is dialing; (2) when *inCall* is accepted, $\Omega_i(X)$ is connected (no difference in not-on-hold or on-hold).

Second, we run an online algorithm for inference. This accelerates the process without waiting for all the signaling messages to come. Upon receiving a new signaling message, we update its pattern incrementally. Once the update is able to validate $\widehat{\Omega}_i(A) \neq \Omega_i(X)$, `spoof` is detected; We stop data

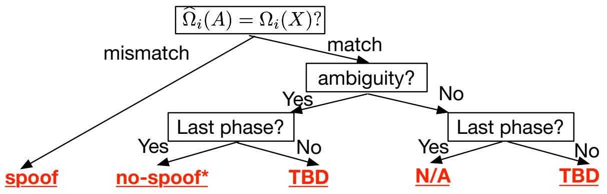


Figure 11: Multi-phase spoofing inference logic.

collection and verification (e.g., stop dialing or hang up this *auCall*). Otherwise, we stop until we receive all the signaling messages. We choose to use the first unique subsequence at runtime in order to complete the spoofing inference early. Note that there are other design options to defer inference and use multiple subsequences (if possible) for reliable inference. We find that, certain signaling message will not be invoked if we do not hang up *auCall* (see §6). We thus add a timer to hang up the call to avoid waiting too long.

Third, our inference decision logic is slightly different. In the training process, the ground truth is known. But in the inference process, we face more uncertainties. As illustrated in Figure 11, our decision tree at each phase have four outputs. (1a) If it does not match any pattern for $\Omega_i(X)$, it stops with '`spoof`'. This is the easiest case. (1b) Otherwise, we consider if the used pattern contains any ambiguity, namely marked with more than one call states. (2a) If no, we stop at '`no-spoof*`' if this is the last phase, otherwise '`TBD`' for next phase. (2b) If yes, we stop at '`N/A`' if this is the last phase, otherwise '`TBD`' for next phase. Note that in 2b, there is alternative aggressive option: we can also mark it as '`no-spoof*`' with lower confidence than the same case in 2a. However, it may generate false-negative results (Ceive says 'no-spoof' when it is a spoof) We choose the current one because we believe that false negative is more damaging. In contrast, marking true negative (`no-spoof`) as N/A may not be a big concern. Given N/A, the callee may stay alert than usual, which is unnecessary when it is not a spoof. Moreover, the callee can be relaxed after learning the call is not ill-intended over the conversation.

Multi-phase verification strategy. Clearly, reducing ambiguity is critical. When one pattern has multiple state labels, one of which matches with $\Omega_i(X)$, it is hard to ensure inference accuracy. Our preliminary study shows that ambiguity is caused by several factors such as indistinguishable call states in one carrier, diversity across unknown carriers (the same pattern means different states in different carriers), user-induced diversity (user setting affecting the pattern). Here, we propose multi-phase verification to tackle it.

First, Multi-phase reduces the N/A likelihood when certain call state is not distinguishable. The N/A probability is the product of those N/A ones at all the phases and greatly reduces with more phases. In this work, we run two-phase verification before and after the call is accepted. Table 3 lists

	No.	Call Scenario	$\Omega_1(A)$	$\Omega_2(A)$
basic	C1	A→B	dialing	conn
	C2	E→B, A is idle	idle	idle
	C3	E→B, A is connected (on-a-call)	conn	conn
	C4	E→B, A is unavailable (i.e, A6)	off	off
advanced	C5	E→B, E (E') made A on a call	conn	conn
	C6	E→B, E (E') is dialing A too	dialed*	dialed*
	C7	E→B, E (E') first dials A and hangs up once B answers the call	dialed*	idle

Table 3: Call Scenarios. 'being dialed' and 'dialing' is indistinguishable in our state inference.

seven typical scenarios. Here, only C1 is no-spoof case. For example, in C2, even when idle is not distinguishable from dialing at phase one, it is detectable as long as conn and idle can be distinguished. This allows us to tolerate coarse-grained call state inference to some extent. We also see that it helps us to combat advanced spoofing strategy. For example, when E dials A in C6 to cheat our verification at the first phase, we still can infer the spoofing at the next phase.

Second, multi-phase verification allows us to combine patterns and get a longer feature vector which combats ambiguity caused by unknown factors. Though A's carrier or other factors are unknown to B, the resulting sequences convey additional information constrained by these unknown factors. Let us use an two-carrier two-phase example to illustrate this idea. Let P_i be the observed pattern while $\Omega_i(X)$ is dialing. Assume that P_i is labelled as `idle` (carrier 1) but `dialing` (carrier 2). Without running more phases, it is believed to be a match with ambiguity and ends with N/A (Figure 11). If we run another phase when $\Omega_{i+}(X)$ is `conn`, we obtain a new observation P_{i+1} which can be `conn` (carrier 1) but cannot be `conn` (carrier 2). Combining both observations, we can infer that $P_i + P_{i+1}$ can not be `dialing` + `conn` for either carrier. We thus ascertain that it is a `spoof`.

In this work, we choose two-phase verification because in the evaluation, it has already achieved 100% accuracy when the spoofing occurs (expect in the stretched attack) using single call action (either VoLTE or CSFB). Theoretically, CEIVE can run more phases as long as each has distinct $\Omega_i(X)$ and π_i (e.g., using hybrid (both VoLTE and CSFB), WiFi calling, VoIP, and other well-designed calling schemes).

4.4 Re-Learning and Other Components

CEIVE also supports learning during the use. This ability is important when our initial training is not sufficient and does not capture key patterns. This also makes CEIVE extensible to new settings (for example, a call from a new carrier which has not been studied before). With re-learning, CEIVE can evolve itself and improve accuracy even if it performs poorly at the start. Re-learning requires user feedback. After one call, CEIVE allows to label this call. We take the same iterative approach in initial training to update our classifiers. Upon a new sample, we need to add this pattern if it never appears, or update the relevant rules if it appears before. If it is consistent with the existing rules, no update is needed. Otherwise, we update its label of call status and re-extract the feature (say, the first distinct subsequence). CEIVE suffers with incorrect samples (e.g., marking a no-spoof as spoof). This may produce wrong ambiguity and mislead CEIVE's inference. Currently, CEIVE works with correct samples only. When a small portion of samples are polluted, we can apply advanced classification techniques (say, majority voting classifiers). This is our ongoing work.

Other triggers for CEIVE. Currently, CEIVE is invoked by any incoming call. It is extensible to other trigger conditions. For example, the user can configure not to run CEIVE when those numbers are from personal contacts, whitelists, call history etc. Billing is another critical factor. In those countries where the user needs to pay extra costs for outgoing calls, CEIVE can be more conservative to make auCalls and even do not run when the call is form one international number or one premium number etc. Note that CEIVE just dials auCalls and hangs up before they get through in most cases, which will not incur extra charges. Moreover, it can work with the existing solutions which mark some suspecting numbers.

What if A also CEIVE-enabled? CEIVE does not require additional support from A. But it should work gracefully in this case. We avoid the chain effect (B calls A, A calls B and into a loop) by allowing at most one active verification test for one number at one time. So even when A calls back to B, B will not a invoke new verification call.

5 IMPLEMENTATION

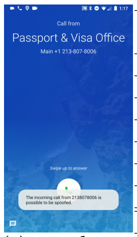
We implement CEIVE on Android smartphones. It is a proof-of-concept prototype addressing three practical implementation issues. First, commodity mobile OS (Android, iOS, etc) does not open permissions to obtain cellular signaling messages. We thus use rooted phones to enable data collection (SIP via TCPDUMP, CSFB/CS signaling via MobileInsight [36]). Second, the current cellular network does not allow another dialing when being dialed. B thus cannot make a call to A while receiving an incoming call request. We prototype CEIVE using a buddy phone B*. B* can be from a family number, a friend or buddy trusted by B. When B makes a call during dialing, B forwards this request and associated information to B*. B* will do it exactly as designed on B and then return results to B. In our implementation, we use Google Firebase [29] to register and obtain buddy services and use the Internet for B-B* communication. Note that the buddy option will not cause any chain effect when both A and B are CEIVE capable. This is because the incoming call will not show up when the phone is dialing or being dialed. In the absence of spoofing, A will not see a request from B*. In the presence of spoofing, A may ask A* to call B* upon receiving the request from B*. This call will not show up at B*, because B* is dialing. Last, cellular networks and Android OS permit two calls but do not allow both active simultaneously. The incoming call is put on hold when B makes another new call, and gets resumed when Ceive ends. This slightly affects user experience. To program voice call services, we use `TELEPHONY_SERVICE`, a system service in TelephonyManager[19] to monitor any incoming call and obtain phone information; We use `ACTION_CALL` in Android Intent to launch a new verification call, which automatically places the prior incoming call on hold; We use `Java reflection` to access the `endCall()` function defined in `ITelephony`. We thus terminate the verification call once having sufficient information for spoofing inference.

6 EVALUATION

We evaluate Ceive in the six aspects: effectiveness against real spoofing, accuracy, extensibility, user friendliness, responsiveness and overhead.

Experiment settings: We assess Ceive in four basic call scenarios and under three advanced spoofing attacks (Table 3). C2-C4 are simple spoofing scenarios where E only fabricates A.ID. C5-C6 are two advanced spoofing attacks where E also manipulates A's state (e.g., being dialed, connected). C7 is a special attack designated against Ceive. E synchronizes his operations to B and A, where E first dials A when dialing B, and hangs up once the call is accepted by B. To amplify damages, we assume A follows E's will (e.g., A will not accept or reject the call and stop the state of being dialed). By default, B is idle before *inCall* comes. We also consider other scenarios where B is in an ongoing call, B is dialing, B is being dialed by someone else when the call comes. B will not receive the call in the latter two cases. There is no difference when B is already on-a-call. We present the results when B is initially idle.

We run experiments in a responsive and controlled manner. All parties (A,B,E) are under our control unless specified. We use 12 Android phones, covering 8 models from Samsung Galaxy S5/S8, Google Pixel XL/2, Nexus 6/6P, LG G4, Xiaomi Mix2 with OSes ranging from 4.4.2 to 8.0.0. We also recruit 12 volunteers (5 local and 7 out-of-states) to act as A only. They use iPhones (6/6s/6p/7/7p) and Android phones. We test with different phone models, and find no phone-specific results, except that B must run Ceive over an rooted Android phone. We run Ceive over VoLTE or CSFB. In VoLTE experiments, we run T-Mobile VoLTE on S5 phones and Verizon VoLTE on LG G3 phones as B and B's buddy. Note that in the US, only T-Mobile and Verizon support VoLTE now (no VoLTE for Sprint; VoLTE restricted in AT&T). On the A's side, we test with all top four US carriers, several single-line landlines, as



The incoming call from 2138078006 is possible to be spoofed.

(a) a real test

A	Basic Spoofing				Advanced Spoofing		
	C1	C2	C3	C4	C5	C6	C7
AT&T CS	N/A*	100%	100%	100%	100%	100%	N/A
T-Mobile CS	100%	100%	100%	100%	100%	100%	100%
T-Mobile VoLTE	100%	100%	100%	100%	100%	100%	100%
Verizon CS	N/A*	100%	100%	100%	100%	100%	N/A
Verizon VoLTE	100%	100%	100%	100%	100%	100%	100%
Sprint CS	100%	100%	100%	100%	100%	100%	100%
CT-Mobile CS	100%	100%	100%	100%	100%	100%	100%
Landline	N/A*	100%	100%	100%	100%	100%	N/A

(b) B running VoLTE (T-Mobile and Verizon)

Figure 12: Effectiveness and accuracy results of Ceive running over VoLTE.

		C1	C2	C3	C4	C5	C6	C7
AT&T	N/A	8/8	0/8	0/8	0/8	0/8	0/8	8/8
	Accuracy	0%	100%	100%	100%	100%	100%	0%
T-Mobile	N/A	7/8	0/8	0/8	0/8	0/8	0/8	7/8
	Accuracy	100%	100%	100%	100%	100%	100%	100%

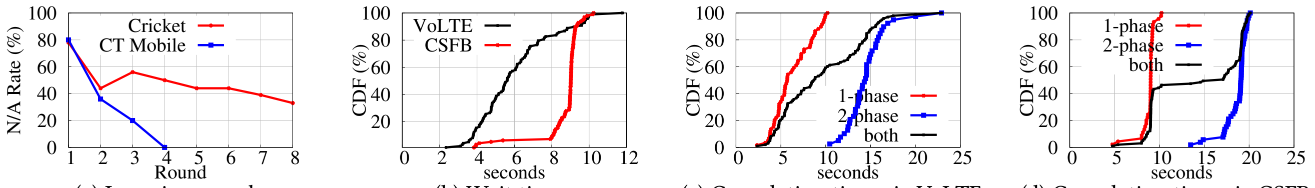
Table 4: Accuracy of Ceive over CS/CSFB.

well as an international roaming carrier and a small US carrier. A runs CSFB/CS and VoLTE if supported. In CSFB tests, we consider B for AT&T or T-Mobile, because the MobileInsight tool [36] does not support 3G CDMA call signaling (in Verizon and Sprint); Ceive runs at most two phases where *auCall* is made before and after the call is accepted.

Effectiveness against real spoofing attacks. We launch the spoofing attack described in §2 towards a Ceive-enabled phone (Pixel 2). Figure 12a shows that Ceive effectively detects `spoof` and completes within 9 seconds while B runs VoLTE in T-Mobile. We validate that it works in all four B's options: VoLTE in T-Mobile and Verizon, CSFB in T-Mobile and AT&T. Remarkably, no other solutions work well. Google's dialer [5] and TrueCaller [51] mislead the callee to believe this number is from 'Passport & Visa Office' (actually, Consulate General of China in Los Angeles). Because this number is a landline out of our control, we cannot run all the attack scenarios (C2-C7). We use the public spoofing service to launch a $E \longrightarrow B$ call faking the ID used in the real scam call. We do not know A's true state and C2/C3/C4 is possible (likely C2). Clearly, Ceive successfully shields against spoofing using the callee-side power only. With Ceive, the victim is able to immediately realize that the incoming caller ID is not trustworthy and likely prevent from the telephony frauds atop.

Accuracy. We further assess its effectiveness in more scenarios. In VoLTE experiments, B uses two carriers and A uses eight carrier-call technology settings. In all tests, Ceive runs with no prior information on A's carrier and call technology. We observe the same accuracy results for both carriers (T-Mobile and Verizon) and combine them in Figure 12b.

Ceive has three outputs: `spoof`, `no-spoof*` and `N/A`. We assess accuracy only in the former two cases and count N/A as the missing rate. It achieves 100% accuracy as long as it infers spoof/no-spoof. It remains 100% effective (true positive) in all spoofing scenarios, except C7 under certain settings.



(a) Learning speed (b) Wait time (c) Completion time via VoLTE (d) Completion time via CSFB

Figure 13: Extensibility and responsiveness results in Ceive.

Note that C2-C6 cover the most common and advanced spoofing attacks. C7 is an stretched attack which has not been observed in use; It requires special efforts to synchronize its manipulation on A. Without such synchronizations, Ceive can possibly detect the spoofing attack. In C7, Ceive outputs N/A in 3 out of 8 basic experimental settings and infers spoof in the rest five settings. Ceive still can keep the users from possible spoofing attacks, as long as the callee stays alert given N/A. However, it implies that the user needs to pay extra efforts when Ceive infers N/A in no-spoof settings (C1). As a matter of fact, Ceive observes the same sequence when C1 or C7 happens. This is why Ceive infers N/A because Ceive adopts a conservative option upon pattern ambiguity (here, the same pattern observed when A is conn or idle). If Ceive enforces an deterministic inference which aggressively converts N/A into no-spoof with high confidence, all N/A results will turn into 100% accuracy in C1 but 0% in C7. As described in §4.3, N/A is more tolerable because the user has no risk for staying alert in case of no spoof. High missing rate in C1 (and in C1 only) are still able to free the users from the spoofing risks, though it is a potential downside to be addressed in the future.

Table 4 shows that Ceive also works well over CSFB. It also achieves 100% accuracy in C2-C6, and faces similar N/A issues in C1 and C7. In C1/C7, it is less effective than VoLTE; It outputs N/A in all the runs when B uses AT&T; When B uses T-Mobile, it can combat ambiguity in only 1 out of 8 settings (T-Mobile VoLTE). This is because CSFB conveys less information than VoLTE. Note N/A is tolerable and Ceive is still effective in the spoofing cases except C7. This indicates that Ceive is ready for wider applicability (as not all 4G carriers support VoLTE to date).

Extensibility. We also assess how Ceive learns and adapts to new scenarios. We examine Ceive's learning speed when applying our algorithm learned from four US carriers and one land-line over two new carriers. CT-Mobile is an international roaming carrier with limited SIP message pattern diversity, and we get 0% N/A rate after 4 rounds (Figure 13a). Cricket Wireless has various SIP message sequences at the dialing state, and also suffers from the same SIP pattern at the idle and connected states. It thus takes much longer learning time, yet still exposed to the risk of failing to infer all cases after convergence (Figure 13a). Note that, this is also caused by N/A in C1 and C7 as well.

User friendliness and responsiveness. Ceive seeks to minimize unnecessary changes and remain friendly to normal users including the callee victim B and the spoofed entity A as well. Ceive largely does well, but still imposes some noticeable (possibly annoying) changes to users.

For B, the obvious change is to prompt the spoofing detection result on the screen, which is expected and needed by B. The second change is related to Ceive's responsiveness. Ceive requires the users to accept the call until it completes the first-phase verification. This induces extra wait time for call answering. We measure the completion time of the first-phase in all the experiments in Figure 13b. Users needs to wait for 4–10 seconds using VoLTE, which is faster than CSFB (mostly, 8-10 seconds). This matches with our experience that CS call setup is slightly slower. When the second phase is needed, Ceive holds the incoming call for 3-10 seconds and then resumes; This might upset B when the call is not malicious (no-spoof).

We further quantify responsiveness in terms of the completion time needed for Ceive. Figure 13c and Figure 13d plot the results using VoLTE and CSFB. we note that, there exists uncertain delay incurred by user operations when the second-phase is required (B must answer the call first). We thus measure the time at B's side between the phone's starting to ring and the final decision, excluding the human delay (i.e., the interval between the first AuCall state being inferred at B and B's answering the call.) We find that Ceive requires only one phase in 60% cases using VoLTE and 40% cases using CSFB. When only one phase is required, it completes within 10 seconds. For most cases (>90%), Ceive finishes within 16 seconds (VoLTE) and 19 seconds (CSFB), up to 23 seconds. Clearly, Ceive yields timely verification. It can alert the user before the telephony fraud take effects (no real loss within tens of seconds).

For A, Ceive has no notifiable changes, if A is not spoofed (the incoming call is made by A). This is because dialing from another party (B or B*) will not be shown up if A is dialing. If A is spoofed, A may notice an incoming call from B or B's buddy. Our user study shows that, A likely has no chance to take this verification call because it ends right after it rings.

Once it starts to ring, Ceive has enough information from call signaling. However, A may call back later once he notices a missed call. This may increase unnecessary calls. But one benefit is to help A realize that A's ID has been spoofed by others when calling back. There are more options to handle this case. For example, we can offer other automated options. such as a recorded voice message indicating that the calls from B or B* is used for verification only, or we can signify the verification from the phone number (e.g., a dedicated number registered for the spoofing verification service provided by Ceive). We treat it as our future work.

Low overhead. We use built-in tools and apps on the phones to measure CPU[9], memory and battery usage (in the Setting panel). We do not notice that Ceive consumes extra CPU, memory, energy when running in the background (without incoming calls). When Ceive is invoked by an incoming call, the incurred overhead is comparable to that of making a call out without Ceive. That is, the overhead is caused by call making. The overhead induced by Ceive is negligible.

7 DISCUSSION

We discuss other possibilities and remaining issues.

Better solution: implementation at the network? Our idea can be implemented at the network as well. It is better if so. The callee's carrier may detect/stop caller ID spoofing by running verifications (more options available inside the network). It can be done even before it rings at the callee, without hurting any user experience. It can even create more signaling for this purpose.

Deployment issues. Deploying Ceive indeed faces several practical issues: It needs a buddy to make an *auCall* when the incoming one rings (no need in other cases); The phone making *auCall* (victim or victim's buddy) must be rooted. These constraints come from the phone OS and chipset vendors. It is possible to relax some: root is not needed in a customized OS (e.g., Android Open Source Project); buddy is not needed with an adjusted verification. Real use may start with selective groups (e.g., seniors who are among the top victims of scam calls).

Possible Downsides. Ceive should conceptually work in any other carrier but its effectiveness depends on signaling realization in various carriers. Given possible carrier-specific customizations, learning over more other carriers is required (using the proposed technique). Another downside is that, this solution may not work when the spoofing call is from a multi-line phone system or any other telephony network where multiple entities share the phone number [54]. As the verification call may reach another entity different from the original caller, state inference may not work unless other information is exposed during call signaling. We have tested with several 800-lines, each of which likely runs a multi-line system. We find that the received signaling sequences do not vary no matter whether A is on a call or being dialed (by our another phone) or we do nothing with A. We gauge that this is because the number is accessible as a whole even when some lines are in use (not all the lines occupied). This matches with our expectation. We note that all cellular networks and most landline carriers are single-line.

New security issues by Ceive. One may concern Ceive is exploited for unintended, malicious usage. For example, an adversary leverages Ceive to launch DoS attacks towards A by making calls to many parties using spoofed A.ID. However, all Ceive's calls are invoked by incoming calls; the attacker already has the capability to make many calls. There is no difference from making many calls directly to A with spoofed caller IDs, thus an non-issue with Ceive.

Spoofing for valid causes. Caller ID spoofing may be used for valid causes, such as anonymity for privacy protection. Our goal is to detect spoofing, regardless of its good or ill intentions. We leave the decision to mobile users on whether to accept or reject the call. We believe that, an alert regarding caller ID spoofing can greatly help those technology-unsavvy people, who are often the target victims of scam calls, to stay alert against malicious caller ID spoofing.

8 CONCLUSION

The paper presents the design, implementation and evaluation of Ceive. Ceive takes a fresh view on cellular-specific operations and low-layer call signaling, in order to differentiate a spoofed caller ID. It thus presents a novel, callee-only solution against caller ID spoofing. It devises various inference techniques to infer the remote caller state, by exploiting an unexplored side channel of 4G networks.

Different from all existing approaches, Ceive is possibly the first effective and practical solution using the callee's capability only. Without requiring any additional infrastructure update or caller-side cooperation, Ceive offers a unique opportunity to take immediate action and combat caller ID spoofing. The defense capability will further improve, as Ceive is being refined and more extensively assessed. Our experience with Ceive also offers a showcase example, where new network security designs can be posed as inference problems and solutions can be devised by applying general machine learning while exploiting deep domain knowledge.

Acknowledgement. We greatly appreciate our anonymous shepherd and reviewers for their constructive comments. This work was partially supported by NSF Grants: CNS-1750953, CNS-1753500 and CNS-1749045.

REFERENCES

[1] 2007. RFC5009: Private Header (P-Header) Extension to the Session Initiation Protocol (SIP) for Authorization of Early Media.

[2] 2015. "Largest IRS Phone Scam Likely Exceeded 450,000 Potential Victims in March". https://www.pindrop.com/irs-phone-scam-live-call_analysis/.

[3] 2015. RFC7462:URNs for the Alert-Info Header Field of the Session Initiation Protocol (SIP).

[4] 2015. Voice over LTE. http://www.gsma.com/technicalprojects/volte.

[5] 2016. Google Phone App. https://play.google.com/store/apps/details?id=com.google.android.dialer.

[6] 2016. Victims lose more than $1 million to China phone scam: Police. http://www.straitstimes.com/singapore/courts-crime/victims-lose-more-than-1-million-to-china-phone-scam-police.

[7] 2017. Chinese callers phish personal information in new phone scam, one person loses over $100k. http://www.straitstimes.com/singapore/chinese-callers-phish-personal-information-in-new-phone-scam-one-man-loses-over-100k.

[8] 2017. Chinese police arrest 118 in scam targeting seniors. http://www.xinhuanet.com/english/2017-09/20/c_136624766.htm.

[9] 2018. CPU Profiler. https://developer.android.com/studio/profile/cpu-profiler.html.

[10] 2018. Fake Call - Fake Caller ID. Mobile app at Google Play and App Store.

[11] 2018. Missed call phone scam still catching Australian mobile users off guard, ACCC says. http://www.abc.net.au/news/2018-02-07/international-missed-call-scam-still-affecting-australians/9396072.

[12] 3GPP. 2011. TS24.007: Mobile radio interface signalling layer 3; General Aspects.

[13] 3GPP. 2017. TS23.272: Circuit Switched (CS) fallback in Evolved Packet System (EPS).

[14] 3GPP. 2017. TS24.008: Mobile Radio Interface Layer 3.

[15] 3GPP. 2017. TS24.229: IP multimedia call control protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); Stage 3.

[16] 3GPP. 2017. TS24.615:Communication Waiting (CW) using IP Multimedia (IM) Core Network (CN) subsystem; Protocol Specification.

[17] 3GPP. 2017. TS24.628: Common Basic Communication procedures using IP Multimedia (IM) Core Network (CN) subsystem.

[18] Android. 2017. Precise Call State. https://android.googlesource.com/platform/frameworks/base.git/+/master/telephony/java/android/telephony/PreciseCallState.java.

[19] Android. 2017. TelephonyManager. https://developer.android.com/reference/android/telephony/TelephonyManager.html.

[20] BBC. 2016. The massive phone scam problem vexing China and Taiwan. http://www.bbc.com/news/world-asia-36108762.

[21] Bloomberg. 2017. Millennials Are Most Likely to Fall for an IRS Scam. https://www.bloomberg.com/news/articles/2017-04-26/millennials-are-most-likely-to-fall-for-an-irs-scam.

[22] Yigang Cai. 2012. Validating caller id information to protect against caller id spoofing. US Patent 8,254,541.

[23] CFCA. 2017. 5 phone scams to watch out for right now - the criminals that are calling you to hack your account. https://www.mirror.co.uk/money/5-phone-scams-watch-out-10748178.

[24] Stanley Taihai Chow, Vinod Choyi, and Dmitri Vinokurov. 2016. Caller name authentication to prevent caller identity spoofing. US Patent 9,241,013.

[25] Federal Trade Commission. 2017. FTC Releases Annual Summary of Consumer Complaints. https://www.ftc.gov/news-events/press-releases/2017/03/ftc-releases-annual-summary-consumer-complaints.

[26] Federal Trade Commission. 2018. Scammers impersonate the Social Security Administration. https://www.consumer.ftc.gov/blog/2018/01/scammers-impersonate-social-security-administration.

[27] Serdar Artun Danis. 2015. Systems and methods for caller ID authentication, spoof detection and list based call handling. US Patent 9,060,057.

[28] Vijay K Garg. 1999. *IS-95 CDMA and CDMA2000: Cellular/PCS systems implementation.* Pearson Education.

[29] Google. 2017. Firebase Projects. https://firebase.google.com/.

[30] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. 2000. FreeSpan: frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 355–359.

[31] IANA. 2017. Session Initiation Protocol (SIP) Parameters. https://www.iana.org/assignments/sip-parameters/sip-parameters.xhtml.

[32] iFeng. 2018. Alert! Phone Scam targeting Chinese from China's Consulates across the US! Someone lost Millions of dollars (in Chinese). http://wemedia.ifeng.com/47830827/wemedia.shtml.

[33] Tao Jiang and Ming Li. 1995. On the approximation of shortest common supersequences and longest common subsequences. *SIAM J. Comput.* 24, 5 (1995), 1122–1139.

[34] KTVB. 2017. Caller ID spoofing on the rise in Ada County. http://www.ktvb.com/article/news/crime/caller-id-spoofing-on-the-rise-in-ada-county/449086755.

[35] Jikai Li, Fernando Faria, Jinsong Chen, and Daan Liang. 2017. A Mechanism to Authenticate Caller ID. In *World Conference on Information Systems and Technologies*. Springer, 745–753.

[36] Yuanjie Li, Chunyi Peng, Zengwen Yuan, Jiayao Li, Haotian Deng, and Tao Wang. 2016. MobileInsight: Extracting and Analyzing Cellular Network Information on Smartphones. In *ACM MobiCom*.

[37] MarketWatch. 2017. Here's how much phone scams cost Americans last year... https://www.marketwatch.com/story/heres-how-much-phone-scams-cost-americans-last-year-2017-04-19.

[38] Hossen Mustafa, Wenyuan Xu, Ahmad Reza Sadeghi, and Steffen Schulz. 2014. You Can Call but You Can't Hide: Detecting Caller ID Spoofing Attacks. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. IEEE, 168–179.

[39] Hossen Mustafa, Wenyuan Xu, Ahmad-Reza Sadeghi, and Steffen Schulz. 2016. End-to-End Detection of Caller ID Spoofing Attacks. *IEEE Transactions on Dependable and Secure Computing* (2016).

[40] The News & Observer. 2017. Scammer using State Bureau of Investigation phone number in fraud scheme. http://www.newsobserver.com/news/local/crime/article160888534.html.

[41] Consulate General of the People's Republic of China in New York. 2017. "Phone Scam Alert". http://newyork.china-consulate.org/eng/lqfw/lsbhyxz/t1486921.htm.

[42] OA Online. 2018. "DOJ warns of telephone scam". http://www.oaoa.com/news/crime_justice/article_a54bf226-0093-11e8-91ba-93e4492d41d7.html.

[43] RFC3261 2002. RFC3261: SIP: Session Initiation Protocol. RFC 3261.

[44] Merve Sahin, Aurélien Francillon, Payas Gupta, and Mustaque Ahamad. 2017. Sok: Fraud in telephony networks. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*. IEEE, 235–250.

[45] ShowCaller. 2017. https://play.google.com/store/apps/details?id=com.allinone.callerid&hl=en.

[46] Jaeseung Song, Hyoungshick Kim, and Athanasios Gkelias. 2014. iVisher: real-time detection of caller ID spoofing. *ETRI Journal* 36, 5 (2014), 865–875.

[47] spoofcard. 2018. Spoofcard Free Spoof Call. https://www.spoofcard.com/free-spoof-caller-id.

[48] Spooftel. 2018. Spooftel Free Caller ID Spoofing Trial. https://www.spooftel.com/freecall/call.php.

[49] New York Times. 2012. Multinational Crackdown on Computer Con Artists. http://www.nytimes.com/2012/10/04/business/multinational-crackdown-on-computer-con-artists.html?_r=0.

[50] Trapcall. 2017. https://www.trapcall.com/.

[51] Truecaller. 2017. https://www.truecaller.com/.

[52] Huahong Tu, Adam Doupé, Ziming Zhao, and Gail-Joon Ahn. 2017. Toward Standardization of Authenticated Caller ID Transmission. *IEEE Communications Standards Magazine* 1, 3 (2017), 30–36.

[53] whoscall. 2017. https://whoscall.com/.

[54] Wikipedia. [n. d.]. Business telephone system. https://en.wikipedia.org/wiki/Business_telephone_system.

[55] Xinhua. 2017. Phone scams targeting NYC Chinese communities exposed. http://www.xinhuanet.com/english/2017-08/10/c_136513524.htm.