بسم الله الرحمن الرحیم


# دروازه فرشتگان:

# یک پروتکل انعطاف پذیر امن اینترنت اشیا جهت ارتباطات ابری–مهی

مهرماه ۹۷

# فهرست مطالب

# فهرست جداول

# فهرست اشکال

# چکیده

ارتباطات و اتصالات همه جا حاضر دیگر یک چالش یا خیال پردازی به شمار نمی آید بلکه یکپارچه سازی افراد و دستگاه ها برای همگرایی قلمرو فیزیکی با محیط های مجازی ساخت دست انسان منتج به تشکیل آرمان شهر اینترنت اشیا شده است. با نگاه دقیق تر به این پدیده، دو واژه تشکیل دهنده آن، اینترنت و اشیا مستلزم شفاف سازی بیشتری می باشند. واژه اشیا در وهله اول شاید مجموعه ایی از هر شی با قابلیت اتصال به اینترنت در نظر گرفته شود اما این واژه نمادی برای اشاره به مجموعه عام تری از موجودیت ها شامل دستگاه های هوشمند، حسگرها، انسان ها، و دیگر اشیا است که از محیط خود، آگاه و قادر به برقراری ارتباط با دیگر موجودیت ها در هر زمان و مکان قابل دست یابی می باشند. رایانش مهی به عنوان یک الگو پردازشی توزیع شده، کلیه مولفه های یک برنامه کاربردی در هردو طرف سرور ابری و تجهیزات لبه ایی مابین سنسورها و سرور ابری را شامل می گردد. به طور معمول اطلاعات تبادل شده در اینترنت اشیا، اطلاعات مهم و حیاتی سناریوهایی همچون مدیریت بحران، سلامت، و مدیریت منابع می باشند. از این رو امنیت تبادل اطلاعات در چنین محیط هایی می تواند تاثیرات مستقیمی بر روی زندگی و نحوه انجام امور روزمره داشته باشد. این شرایط فرصتی بزرگی را برای هکرها فراهم می سازد تا امنیت و حریم شخصی چنین محیط هایی را نقض کنند که منجر به نتایج جبران ناپذیری خواهد شد. ذکر این نکته ضروری است که طراحی امنیتی پیشنهادی برای محیط های اینترنت اشیا نه تنها بایستی بتواند جلوی خطرات احتمالی از فضای عمومی اینترنت را بگیرد بلکه باید از کلیه اجزا سالم و سازنده در شبکه در مقابل اجزا مخرب محافظت کند. در این پروژه، ما طراحی و اجرای "دروازه فرشتگان" را ارایه کرده ایم. دروازه فرشتگان یک پروتکل انعطاف پذیر و امن جهت ایمن سازی ارتباطات ابری-مهی در بستر اینترنت اشیا می باشد. پروتکل پیشنهادی شامل چهار بخش تشکیل دهنده می باشد: بخش امنیتی، بخش بلوک زنجیره ایی، بخش مسیریابی امن، و بخش به روزرسانی انرژی-محور بلادرنگ. در بخش اول، مکانیسم های احرازهویت و امضای دیجیتال به همراه رمزنگاری های متقارن و نامتقارن استفاده شده است. در بخش بعدی، مکانیسم بلوک زنجیره ایی به منظور تایید صحت زنجیره درخواست های سلسله مراتبی هر موجودیت به کار گرفته می شود. بخش مسیریابی امن، نه تنها امکان مسیریابی از هر گره به سرور ابری را حتی در صورت قطع احتمالی ارتباطات نظیربه نظیر فراهم می کند، بلکه از وقوع حملات ربودن جلسات در سطح شبکه خودداری می کند. در آخر، بخش به روزرسانی وظیفه همگام سازی امن گره ها با سرور ابری با در نظر گرفتن بهره وری انرژی را بر عهده دارد. نتایج آزمایشگاهی نشان می دهد که راهکار پیشنهادی بستر امنی جهت ارتباطات ابری-مهی با حداقل سربار پردازشی و حداکثر انعطاف پذیری با امکان تعیین و تغییر دلخواه اندازه کلیدها و الگوریتم های رمزنگاری را فراهم می کند. همچنین نتایج به دست آمده مقاومت راهکار پیشنهادی در برابر حملات شناخته شده برعلیه اصول مرجع سه گانه امنیتی یعنی محرمانگی، یکپارچگی و دسترس پذیری را تایید می کند.

# 1 مقدمه

سال هاست که در پیاده سازی برنامه ها، سرویس ها و پلتفرم های مختلف (که تبادل پیام بین حداقل دو گره وجود داشته باشد) نیاز به یک پروتکل امنیتی بسیار دقیق ، کامل و بی نقص احساس می شود. اهمیت داده ها از یک سو و اطمینان از عدم جعل و شنود داده از سوی دیگر هر روز مهم تر و پر اهمیت تر می شود تا جایی که عملا بدون پروتکل های امنیتی، امکان ایجاد حریم شخصی و امنیت کاربران (گره ها) امکان پذیر نخواهد بود. با گسترش روزافزون علوم و پیدایش فناوری های جدیدی همچون اینترنت اشیا [1] که با زندگی امروز بشر آمیخته و عجین شده، نیاز به امن بودن تبادلات پیام ها و اطلاعات بین گره ها در حالت های متمرکز (سرور ابری-گره) و نامتمرکز (گره-گره) بسیار حائز اهمیت است. تبادل اطلاعات در این دستگاه ها گاها بسیار حیاتی اند و چنانچه شنود یا جعل گردند می توانند اثرات و مشکلات حاد و جبران ناپذیری به بار آورند. علاوه بر امنیت پیام ها و اطلاعات ارسالی و دریافتی، وجود کنترل کننده ها و پاسخ دهنده های مناسب و هماهنگ جهت ایجاد پایداری و جلوگیری از برخی حملات از جمله انکار خدمات و زیر ساخت اهمیت و جایگاه ویژه ایی دارد. این کنترل کننده ها نیز می بایست به کمک الگوریتم ها از قواعد منطقی پیروی کنند تا وظیفه خود رابه نحو احسن انجام دهند. چنانچه این مجموعه به صورت هماهنگ و واحد با یکدیگر کار کنند، می توان به امنیت قطعی و کامل اطلاعات تبادل شده مطمئن شد.

همچنین با پیشرفت فناوری ها، بیشتر قریب به تمام پلتفرم ها و سرویس ها از شی گرایی و زبان های برنامه نویسی متنوع استفاده می کنند، لذا پروتکل ارائه شده می بایست بتواند کلیه ی نیازهای موجود را پاسخ دهد. به عبارت دیگر، از فناوری های موجود به گونه مستقل از پلتفرم و زبان برنامه نویسی پشتیبانی کند و خود پروتکل بتواند به صورت یک $SDK$[2] سازگار با هر زبان برنامه نویسی و سیستم عاملی قابل بکارگیری باشد. از طرفی در سیستم ها، پلتفرم ها و سرویس های امروزی انواع معماری های تبادل داده و احراز هویت وجود دارد که پروتکل پیشنهادی می بایست بتواند با انواع مختلف این معماری ها بدون نقص و کاستی کار کند تا به کارآمد ترین پروتکل تبادل اطلاعات امن شده بدل شود. برای نمونه، در انواع اتصال های متمرکز، می توان به حالت های مستقل از وضعیت [3] و بر اساس وضعیت[4] یا مکانیسم های اتصال های نیمه دوطرفه [5] و دوطرفه[6] ، اتصالات بلادرنگ [7] ، گردآوری کوتاه [8] و گردآوری بلند [9] ، احراز هویت بدون جلسه [10] و مبتنی بر جلسه[11] اشاره کرد که بایستی توسط این پروتکل پشتیبانی گردد و قابلیت به کارگیری در کلیه این حالات را داشته باشد.

---

[1] Internet of Things (IoT)
[2] Software Development Kit
[3] Stateless
[4] Stateful
[5] Half Duplex
[6] Full Duplex
[7] Real-time
[8] Short Polling
[9] Long Polling
[10] Session-less
[11] Session-based

دروازه فرشتگان، نام پروتکل پیشنهادی به منظور انتقال اطلاعات به صورت امن شده در بستر شبکه و اینترنت اشیا می باشد. این پروتکل مجموعه ایی از قواعد، الگوریتم، کنترل کننده ها و پاسخ دهنده ها است که به صورت هماهنگ و یکپارچه با یکدیگر فعالیت می کنند تا پیام های تبادل شده بین گره ها با سرور ابری[12] و گره با گره[13] در بستر شبکه به هر صورتی که باشد به طور کاملا امن و بدون امکان شنود و دست کاری تبادل گردد.در ادامه، ویژگی های و معماری های مختلف این پروتکل به همراه جزییات شرح داده شده است.

## 2 ویژگی های پروتکل پیشنهادی

از ویژگی های اصلی و ویژه پروتکل پیشنهادی به موارد زیر می توان اشاره کرد:

1) امکان تایید صحت داده های رمزنگاری شده با استفاده از امضا الکترونیکی داده ها (امکان اتصال به شبکه زنجیره بلوک [14])
2) عدم پذیرش کلیه پکیج های پیام (بسته های داده) که صحت آن ها تایید نشده، تکراری یا قدیمی هستند
3) پشتیبانی از مکانیسم $JSON$ برای تبادل داده هایی که به صورت شی و لیست هستند
4) پشتیبانی از مکانسیم $Base64$ برای ارسال داده هایی که به صورت باینری و فایلی هستند
5) رمزنگاری کامل و غیرقابل جعل، شنود و شکسته شدن به خاطر معماری خاص امنیتی آن
6) قابلیت پیاده سازی بسیار آسان در کلیه برنامه ها، زبان های برنامه نویسی و سیستم عامل ها
7) کاملا منبع باز، رایگان و قابلیت ارتقا توسط هر شخص یا سازمان برای استفاده های اختصاصی
8) کاملا مستقل از پلتفرم و زبان برنامه نویسی
9) قابلیت پیاده سازی و به کارگیری در کلیه دستگاه ها و گره های اینترنت اشیا
10) قابلیت پیاده سازی و به کارگیری در برنامه نویسی های موازی و برنامه های پردازش موازی
11) قابلیت پیاده سازی و به کارگیری در شبکه ها و سیستم های خوشه ایی [15] و ابری [16]
12) پشتیبانی از کلیه حالات اتصال شبکه و معماری داده ایی شامل بدون وضعیت، مبتنی بر وضعیت، بدون جلسه، مبتنی بر جلسه، بلادرنگ، مبتنی بر درخواست [17]، سوکت، HTTP و غیره.
13) بسیار سبک، کم حجم و با سربار پردازشی حداقل
14) امکان تعیین و تغییر دلخواه نوع الگوریتم های اصلی رمزنگاری و اندازه کلیدها

---

[12] Cloud Server to Peer
[13] Peer to Peer
[14] BlockChain
[15] Cluster
[16] Cloud
[17] On-demand

## 3 مقاومت در برابر حملات

پروتکل پیشنهادی در برابر کلیه حملات و آسیب پذیری های زیر مقاوم است:

1) حمله Indistinguishability Chosen Plaintext Attack (IND-CPA)
2) حمله Indistinguishability Chosen Ciphertext Attack (IND-CCA)
3) آسیب پذیری کشف رمز با سرقت کلیدهای ثابت [18]
4) حمله Known Plaintext Attack (KPA)
5) حمله Replay Attack
6) حمله Relay Attack
7) حملات منع دسترسی (Denial of Service Attack)
8) حملات مرد میانی [19] شامل 3-Subset Attack و Biclique Attack
9) حملات جعل و Spoofing (به خصوص Device ID spoof)
10) حملات سرقت جلسه (Session Hijacking) تا حد بسیار بالا بدون از دست دادن دادن قابلیت دردسترس پذیری [20]
11) حملات قطع دسترسی (Null Route) تا حد زیاد
12) حملات نقض حریم شخصی (Privacy Violation) برای کشف طرف های درگیر
13) حملات Adaptive و Non-Adaptive که انواع حملات Chosen Plaintext Attack هستند
14) حملات Length Extension به دلیل استفاده از امضاهای دیجیتالی
15) حملات Adaptive و Non-Adaptive که انواع حملات Chosen Ciphertext Attack هستند
16) حمله Distinguishing Attack
17) حمله Stream Cipher Attack
18) حمله Correlation Attack
19) حمله Brute Force Attack
20) حمله Dictionary Attack
21) حمله Oracle Attack

---

[18] Static Key
[19] Man in the Middle (MitM)
[20] Availability

# 4 پروتکل پیشنهادی

## 4.1 معماری ابری (سرور ابری – گره)

### 4.1.1 درخواست احراز هویت (Pre-Authentication)

این قسمت پیش نیاز شروع ارتباطات بین سرور ابری و گره ها است. قبل از هر گونه تبادل پیام، بایستی عملیات احراز هویت به طور کامل اجرا شود تا علاوه بر اطمینان از اینکه کاربر معتبر و واقعی این جلسه را ایجاد کرده، قسمت هایی همچون $Token$ و $Chain\ Signature$ تخصیص داده شوند. در این مرحله که ابتدای احراز هویت می باشد مانند اکثر مکانیسم های تایید هویت، لازم است تا شناسه ایی جهت ارسال کد یا تماس و دریافت آن ارسال گردد. بسته به مکانیسم ها و طراحی های مختلف نرم افزاری تعریف شده توسط طراح سیستم، این قسمت می تواند اطلاعات مورد نیاز آن احراز هویت را از ورودی دریافت کرده و نسبت به آن اطلاعات در سرور ابری بررسی های لازم را انجام دهد.

چنانچه حساب کاربری یا شناسه دستگاه وارد شده صحیح بود، عملیات صحت سنجی مانند ارسال کد به شماره تلفن یا مکانیسم های مشابه با هر نوع رمز یک بار مصرف [21] اجرا می گردد به طوری که در مرحله تکمیل احراز هویت چناچه کاربر یا گره آن مقدار خواسته شده را صحیح وارد کند عملیات احراز هویت تکمیل می گردد و $Token$ به آن گره تخصیص خواهد یافت و به مرحله بعدی، ارسال و دریافت اطلاعات می رود. بعد از تایید هویت اولیه، مقدار $Handler$ برای مرحله تکمیل احراز هویت از سمت سرور ابری به گره ارائه می گردد و $Handler$ با مقادیر لازم بعدی جهت احراز هویت استفاده خواهد شد.

### 4.1.2 تایید نهایی احراز هویت (Post Authentication)

پس از مرحله درخواست احراز هویت (Pre-Authentication) نوبت به مرحله تایید نهایی درخواست هویت می رسد. لازم به ذکر است در صورت تایید مشخصات و درخواست ها در محله اول توسط سرور ابری، مرحله دوم یا نهایی اجرایی می گرد. همان طور که در قبل اشاره شد این پروتکل طوری طراحی گردیده است که در گره های اینترنت اشیا مورد استفاده قرار گیرد. احراز هویت در این پروتکل می تواند به دو حالت بدون جلسه [22] و با جلسه [23] انجام گیرد. نمونه حالت اول استفاده از شناسانگر دستگاه ($DeviceID$) و نمونه حالت دوم کاربرانی که از حساب های کاربری مجزا استفاده می کنند است. همچنین در حالت با جلسه، چنانچه یک کاربر (به صورت مداوم) فقط از یک دستگاه استفاده کند حالت احراز هویت تک کاربره [24] و اگر کاربر از چند دستگاه همانند اپلیکیشن ها و نرم افزارهای فضای مجازی استفاده کند یا یک دستگاه

---

[21] One-Time Password (OTP)
[22] Session-less
[23] Sessionful
[24] Single User

چندین کاربر داشته باشد، حالت احراز هویت چند کاربره[25] می باشد که تمامی این حالات توسط پروتکل پشتیبانی شده و قابلیت اجرایی دارد.

در مرحله نهایی سازی احراز هویت، مقدار $Handler$ درخواست در پاسخ سرور در مرحله اول احراز هویت می بایست استفاده گردد تا مشخص شود همان دستگاهی که درخواست احراز هویت را صادر کرده به مرحله نهایی سازی احراز هویت وارد شده است. در این مرحله، اطلاعات تکمیلی یا تایید هویت ثانویه مانند کد پیامک شده به سمت سرور ارسال می گردد. چنانچه احراز هویت اولیه و اطلاعات تکمیلی ثانویه صحیح و کامل به سرور ارایه شود، مقدار $Token$ به گره ارائه می شود. بعد از این مرحله، گره بایستی پکیج داده به علاوه $Token$ و زنجیره امضا $(Chain\ Signature)$ داده های ارسالی را به سمت سرور ارسال کند. در غیر این صورت، سرور این احراز هویت و جلسه را ابطال کرده و می بایست عملیات احراز هویت از ابتدا انجام گیرد. بدیهی است تا احراز هویت مجدد سرور درخواست ها را با پیام مناسب رد می کند.

از طرفی دیگر چنانچه گره در هر درخواست پاسخی از سرور دریافت نکرد یا با خطاهای عمومی مواجه شد، بایستی بتواند پکیج را باز طراحی و مجددا ارسال نموده و نسبت به خطایی دریافتی از سرور رفتار صحیحی انجام دهد.

## 4.2 معماری مهی (گره – گره)

### 4.2.1 مسیریابی هوشمند (Smart Routing)

چنانچه گره ها بخواهند در بستر اینترنت اشیا از وجود یکدیگر مطلع گردند و تبادل داده غیر متمرکز با یکدیگر داشته باشند بایستی مکانیسم مسیریابی هوشمند و امنی ارایه گردد که گره ها را به سرور ابری برای مسیریابی غیر متمرکز معرفی کند و امن سازی پیام های تبادلی بین گره ایی را انجام دهد.

در راستای نیل به اهداف فوق و اعلان کلیدهای تبادلات غیر متمرکز محلی لازم است بعد از اتمام مرحله احراز هویت، مرحله اعلان مسیرها انجام بپذیرد و هر گره موظف است اطلاعات را از خود ایجاد و به سرور اعلان کند. در این مرحله بایستی جهت اعلان به سرور مقدار فیلد $Request$ برابر با کلمه رزرو شده $Set\ Route$ یا هر کلمه ایی که به این عنوان لحاظ شده، باشد. همچنین بایستی ارسال اطلاعات در در قالب جدول گره (Peer Table) (جدول 1-1) در فیلد $Data$ برای سرور ابری ارسال گردد. چنانچه این اعلان توسط سرور ابری مورد قبول واقع شود، پاسخ مناسبی به گره ارائه می گردد. بعد از آن هر زمان هر گره ایی قسمتی یا تمامی این اطلاعات را تغییر دهد یا ناچارا اطلاعات تغییر بکند، به همین صورت جدول گره های جدید به سرور ابری اعلان می گردد و در قالب به روزرسانی به دیگر گره ها این تغییرات در جدول بروزرسانی شده توسط سرور ابری اعلان می شوند.

[25] Multi User

جدول 1-4 نمایی از جدول گره *(Peer Table)* که در هر گره ذخیره می گردد

| Label | Endpoint | MyID | PeerID | PKey |
|---|---|---|---|---|
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

پاسخ سرور ابری به گره ها ، جدول حاوی کلیه گره های مجاز با $PeerID$ های صحیح و $MyID$ های صحیح نسبت به $PeerID$ ها می باشد. لازم به ذکر است که این جدول برای هر گره بر اساس $PeerID$ ها و $MyID$ ها منحصر به فرد ایجاد می گردد. به عبارت دیگر، جدول گره هر گره متفاوت با بقیه گره ها و منحصر به همان گره است و برای دیگر گره ها قابل استفاده نیست. ستون های جدول فوق در ادامه توضیح داده شده اند:

$Label$: نام یا شناسه معرف هر گره که می تواند از قبل تعیین شده باشد.

$Endpoint$: آدرس اینترنتی یا درون شبکه ایی برای اتصال و دسترسی به آن گره که می بایست از هر گره دیگری قابل دسترسی باشد.

$PeerID$: در ارتباطات بین گره ایی، فیلد PeerID جایگزین فیلد DeviceID در ارتباطات گره با سرور ابری می گردد که عملکردی مشابه DeviceID دارد ولی به روش متفاوتی ایجاد می گردد و بستگی به جدول Peer Table که گره دریافت می کند دارد.

MyID: مقدار MyID، متناظر با PeerID که گیرنده پیام درخواست است قرار می گیرد و توسط تابع $OR_{enc}$ (رابطه 15) رمزنگاری می گردد.

$Peer\ Pkey$: هر گره یک زوج کلید برای رمزنگاری نامتقارن $RSA$ ایجاد می نماید. کلید خصوصی را نزد خود و کلید عمومی را تحت عنوان $Peer\ PKey$ در اعلان به سرور ارسال می کند که نقش بسیار مهم و حیاتی دارد.

جدول کامل $Peer\ Table$ می تواند توسط هر گره با ارسال رشته رزرو شده $Get\ Route$ یا $Route$ دریافت گردد. همچنین با توجه به منحصر به فرد بودن جدول گره، چنانچه گره ایی آلوده یا سرقت گردد، امکان جعل ارتباط و شناسه گره ناممکن می باشد. تنها آسیب پذیری غیر قابل اجتناب، فاش شدن آدرس های اتصال هر گره یعنی $Endpoint$ ها در صورت سرقت فیزیکی یگ گره می باشد که خطر حملات قطع دسترسی $DoS$ و قطع اتصال علیه گره دیگری را افزایش می دهد.

بعد از انجام کلیه مراحل اعلان و دریافت جدول گره ها، هر گره قابلیت تبادل اطلاعات با گره دیگر را داراست و این ارتباطات در بستری امن و مقاوم در برابر کلیه حملات ذکر شده انجام می پذیرد. همچنین سرقت فیزیکی و مهندسی معکوس یک گره کل اکوسیستم را دچار خطر نمی کند.

### 4.2.2 ارسال پکیج (Package Forwarding)

با وجود ارتباطات غیر متمرکز محلی بین گره ایی می توان مکانیسمی طراحی کرد تا از این ارتباطات برای حفظ و ارتباط گره ها به سرور ابری استفاده کرد که به اصطلاح Package Forwarding نامیده می شود. مکانیسم پیشنهادی می تواند از وقوع حملات منع دسترسی و فیلیترینگ اتصال جلوگیری کرده و اصطلاحا فیلترینگ را دور[26] بزند تا بتواند پیام خود را به دیگر گره ها رسانده و درخواست Forwarding از آن ها بکند. در این موقعیت گره واسط حکم یک Proxy را دارد که پیام را به صورت کاملا امن شده و بدون امکان دستکاری و شنود و بدون هیچ درکی از محتوای آن، به سرور ابری رسانده و پاسخ سرور را به گره درخواست دهنده می رسانند. این مکانیسم از قطع دسترسی های احتمالی یا عمدی که توسط هکرها یا ارائه دهنده های سرویس شبکه اعمال می گردد می تواند تا حد زیادی عبور کند.

همچنین، این مکانیسم می تواند پس از چندین بار تلاش اتصال ناموفق به سرور توسط گره به صورت خودکار فعال گردد و تا رفع نشدن مشکل از گره یا گره های همسایه دیگر جهت تبادل با سرور استفاده کند. در این مکانیسم از ترکیب معماری سرور ابری-گره و گره-گره بدین شرح استفاده می گردد: پکیج رمز شده و کامل که قرار است به سرور ارسال گردد، به طور مستقیم در فیلد $Data$ پکیج بعدی که می بایست به گره دیگر ارسال شود قرار می گیرد و عبارت رزرو شده $Forward$ نیز در فیلد $Request$ قرار داده می شود. وقتی پکیج به گره واسط رسید، آن گره بدون در نظر گرفتن محتوی درون $Data$ ، آن را برای سرور به طور مستقیم و بدون هیچ عمل پردازشی اضافی ارسال می کند. به همین ترتیب، پاسخ دریافتی از سرور به عنوان پاسخ درخواست گره مبدا توسط گره واسط به گره مبدا باز گردانده می شود.

در این مکانیسم، گره ارتباط خود با سرور ابری را از طریق گره یا گره های همسایه به سمت سرور ابری هدایت می کند به طوری که هیچ داده ایی از گره مبدا درمسیر و برای گره یا گره های واسط قابل درک و پردازش نیست و ارسال اطلاعات به طور امنی به سمت سرور صورت می پذیرد.

## 4.3 ارتباطات گره ها با سرور ابری – قسمت درخواست

در این قسمت پکیج استفاده شده دریک درخواست ارسالی از گره ها به سرور ابری معرفی شده و تمامی فیلدهای آن همان طور که در شکل 1.1 به نمایش درآمده است به تفصیل در ادامه توضیح داده شده اند.

[26] Bypass

| $Request$ | $Data$ | $Signature$ | $Server\ Salt$ | $Device\ ID$ | $Time_{Req}$ $/Client_{Salt}$ | $Seq\ \#$ | $Token$ | $Chain\ Signature$ |
|---|---|---|---|---|---|---|---|---|

شکل 1-4 پکیج استفاده شده در ارتباطات گره با سرور ابری- قسمت درخواست

Request: مقدار این فیلد در ابتدای احراز هویت و ثبت نام گره می بایست معادل با رشته رزرو شده ی PreAuth یا هررشته ثابت که به این عنوان استفاده می گردد باشد. در واقع، Request دربردارنده درخواست استفاده از منابع ابری می باشد.

Data: این فیلد شامل اطلاعات خام (رمزنگاری نشده) است که می تواند به فرمت JSON یا Base64 می باشد که با استفاده از تابع رمزنگاری $R_{enc}$ مطابق رابطه (1) رمزنگاری می گردد:

$$R_{enc} = AES_{CBC}(Input, IV_S, KEY_R) \quad (1)$$

که در آن $AES_{CBC}$ الگوریتم رمزنگاری متقارن AES در مد CBC ، $Input$ همان اطلاعات ورودی، $IV_S$ بردار اولیه استفاده شده در الگوریتم رمزنگاری است که به صورت ثابت[27] تعریف می گردد و $KEY_R$ کلید چرخشی متغیر قابل استفاده در $AES_{CBC}$ می باشد که از ترکیب کلید ثابت $KEY_S$ و مقدار تصادفی $S_s$ حاصل می گردد که در ادامه توضیح داده خواهد شد. لازم به ذکر است مقادیر $KEY_S$ ،$IV_S$ و $PKEY$ که همان کلید عمومی سرور ابری است به طور ثابت در کلیه گره ها وجود دارد. این مقادیر به همراه کلید خصوصی سرور ابری، $PrivKEY$ به طور ثابت در سرور وجود دارند.

$Signature$ : این فیلد دربردارنده امضا منحصر به فرد هر درخواست است که نتیجه تابع درهمساز[28] $C_{Sig}$ می باشد و به جهت حفظ یکپارچگی اطلاعات مبادله شده بین گره و سرور ابری به کار می رود. تابع $C_{Sig}$ طبق رابطه زیر تعریف می گردد:

$$C_{Sig} = Hash(S_s + Date\ ('Y') + Request + Data + DeviceID + Token + Seq\# + Time_{Req}, S_s) \quad (2)$$

که در آن $Date\ ('Y')$ نشان دهنده سال حاضر است و دیگر پارامترها در ادامه توضیح داده خواهد شد. این امضا از ترکیب داده های رمزنگاری نشده با سال حاضر،بازه مورد تایید و برخی پارامترهای دیگر است که آن را بی نیاز از رمزنگاری می کند زیرا پارامترهای تشکیل دهنده پس از امضا غیرقابل کشف می باشند. همچنین داده هایی مانند $Time_{Req}$، $Seg\#$ ، $Request$ و $Chain\ Signature$ آشکارکننده و حیاتی نیستند و غیر قابل تغییر و دستکاری اند. این عدم رمزنگاری به نوبه خود موجب کاهش مصرف انرژی و سربار پردازشی می گردد. چنانچه کوچک ترین تغییری در اطلاعات مبادله انجام

---

[27] Static

[28] Hash

گیرد، تایید آن با امضا غیر ممکن شده و درخواست توسط سرور مورد قبول واقع نمی شود. همچنین تابع درهمساز $Hash$ به شکل زیر تعریف می گردد:

$$Hash = \begin{cases} rot13(Base64(SHA256(Base64(Input) + MD5(Salt)))) & if \quad lenght(salt)\ is\ even \\ rot13(Base64(SHA256(SHA1(Salt) + Base64(Input)))) & if \quad lenght(salt)\ is\ odd \end{cases} \tag{3}$$

$Server\ Salt$: برای تولید این فیلد، ابتدا گره یک مقدار بسیار تصادفی[29] را ایجاد می کند که آن را $S_s$ می نامیم. سپس، $S_s$ با استفاده از الگوریتم رمزنگاری RSA و کلید عمومی سرور ابری $(PKEY)$ مطابق رابطه (4) رمزنگاری می گردد.

$$RSA = RSA_{PK}\ (Input) \tag{4}$$

بایستی توجه داشت که طول $S_s$ متناسب با طول کلید در نظر گرفته شده الگوریتم رمزنگاری متقارن باشد. $S_s$ به دلایل زیر اهمیت بسیار زیادی دارد:

1) متغیربودن آن باعث می گردد کلید ایجاد شده برای هر درخواست منحصربه فرد و متفاوت با درخواست قبلی آن باشد. این ویژگی باعث جلوگیری از وقوع بسیاری از حملات و سواستفاده ها از آسیب های رمزنگاری با کلید ثابت می گردد.
2) از آن جایی که $S_s$ به عنوان یک کلید منحصربه فرد و مشابه یک OTP[30] غیرقابل حدس عمل می کند (به دلیل متغیر و بلند بودن طول آن)، از وقوع حملات Man in the Middle (MitM) شامل Reply attack و Relay attack نیز جلوگیری می کند.
3) از آن جایی که $S_s$ شامل اعداد، حروف بزرگ و کوچک و کاراکترها می باشد، از وقوع مشکل Collision جلوگیری می کند.

مقادیر $S_s$ در سرور برای مدت کافی به طور مثال تا زمان انقضای درخواست با توجه به حداکثر زمان Timestamp هر درخواست به صورت جدول Hash ذخیره و حفظ می گردد تا از این که پیامی تکرار یا هدایت غیرمجاز شده باشد جلوگیری کند. بعد از گذشت حداکثر بازه زمانی مجاز درخواست (به طور مثال 24 ساعت) این مقادیر می توانند با استفاده از یک زمانبند از سرورحذف گردند. بدین ترتیب سرور، درخواستی با وقفه بیش از بازه زمانی مجاز Timestamp را رد کرده و پردازش نخواهد کرد. این مکانیسم از وقوع بسیاری از حملات Man in the Middle ، تخریب داده یا هدر دادن منابع سرور همچون DoS[31] جلوگیری می کند. علاوه بر این، سرور با شمارش درخواست های نامعتبر می تواند گره ها یا آدرس IP های

---

[29] Highly Random
[30] One Time Password
[31] Denial of Service

مخرب را تشخیص و با دیواره آتش محدود کند. زیرا هر گره ایی که زمان درخواست آن کمتر یا بیشتر از بازه زمانی مورد قبول تنظیم شده باشد، پیغام خطای مناسب را دریافت و با تنظیم ساعت آن درخواست را تصحیح می کند و متعاقبا درخواست های بعدی آن گره هم تصحیح خواهند شد مگر آن که به طور قطع این گره توسط هکر کنترل یا گره مخرب باشد.

$Device\ ID$: این فیلد نشان دهنده شناسه سخت افزاری دستگاه می باشد که با توجه به مشخصات سخت افزاری دستگاه بدست می آید. به طور معمول مقدار $Device\ ID$ در طراحی ها ثابت در نظر گرفته می شود ولی در صورتی که تمایل طراح به ثابت بودن آن برای همیشه است پیشنهاد می گردد از الگو زیر استفاده شود:

$$DeviceID = Hash\ (MainboardID \oplus "123ABC") \qquad (5)$$

که در آن $Device\ ID$ به صورت امضا یا درهم سازی شده از ترکیب شناسه سخت افزاری (به طور مثال مادربورد) و یک مقدار ثابت می باشد. همچنین بسته به طراحی نرم افزاری که انجام می شود $Device\ ID$ می تواند شناسه جلسه[32] باشد که در هنگام احراز هویت ثبت می گردد که در این حالت می بایستی این مقدار بعد از احراز هویت موفقیت آمیز در حافظه ذخیره گردد.

$Time_{Req}/Client_{Salt}$: این فیلد می تواند نشاندهنده زمان حال حاضر[33] درخواست گره $(Time_{Req})$ یا شمارنده کل درخواست های ارسالی $(Client_{Salt})$ باشد. در معماری پیشنهادی از زمان حال حاضر استفاده گشته ولی معماری های دیگر می توانند مستقل از زمان و بر اساس پارامتر دیگری همچون شمارنده کل درخواست ها طراحی گردند.

$\#Seq$: یک شناسه عددی است که در پکیج درخواست قرار داده می شود تا اگر تعدادی از درخواست ها به صورت موازی[34] ارسال شوند توالی پاسخ های دریافت شده معین گردد.

$Token$: همانند $S_s$ که یک مقدار تصادفی تولید شده توسط گره می باشد، $Token$ یک مقدار تصادفی تولید شده توسط سرورابری به جهت اطمینان از احراز هویت گره درخواست کننده است. در مرحله ابتدایی (PreAuth) که هنوز احراز هویتی صورت نگرفته این فیلد به طور خالی ارسال می گردد. در مرحله بعد، PostAuth، سرور $Token$ را تولید کرده و به گره اعلان می دارد که از این پس مورد استفاده قرار خواهد گرفت، بنابراین بایستی در حافظه گره ذخیره گردد. زمانی که طراح سیستم $Device\ ID$ را ثابت در نظر بگیرد و از هیچ مکانیسمی همچون DH[35] برای تبادل کلید جلسه استفاده نکند ممکن است در معرض وقوع حمله جعل جلسه[36] قرار گیرد. وظیفه Token جلوگیری از وقوع چنین حالت هایی می باشد و این اطمینان را می دهد که دستگاه یا جلسه ایی که از آن درخواست ارسال می گردد قبلا احراز هویت شده است. گره بایستی

---

Session Identifier[32]
Time Stamp[33]
Concurrent[34]
Diffie–Hellman[35]
Spoofing Session[36]

پیامی به جهت اطمینان از دریافت $Token$ به سرور ارسال و آن را مطلع کند. برای این منظور، تابع $Tokenize$ امضا آخرین $Token$ دریافتی را بر اساس رابطه (6) به سرور ارسال می کند.

$$Tokenize = Sha1\ (Token + S_s) \quad (6)$$

که در آن $Sha1$ یک تابع درهم ساز می باشد.

$Token$: می تواند به دو حالت وجود داشته باشد: ایستا و پویا. در حالت اول $Token$ ثابت بوده و یک بار در مرحله PreAuth مقداردهی می گردد و پس از آن از سمت سرور خالی ارسال می گردد. در حالت دوم، سرور می تواند $Token$ را به صورت بازه ایی یا در هر پیام تغییر دهد تا از هویت دستگاه و عدم جعل $DeviceID$ اطمینان بیشتری حاصل کند، اما بازه های کوتاه سربار بیشتری خواهد داشت. چنانچه سرور $Token$ جدیدی را در پاسخ درخواست ارایه کند، آن گاه چگونه می تواند از دریافت آن توسط گره اطمینان حاصل کند و در پیام های بعدی انتظار آن را داشته باشد؟ حال اگر به هر دلیلی همچون قطع اتصال اینترنت، گره $Token$ جدید را دریافت نکرده باشد، گره همچنان با $Token$ قبلی خود درخواست جدید را ارسال می کند که سرور ابری آن درخواست را نامعتبر تلقی می کند و عملیات احراز هویت مجدد الزام می گردد. در صورت تکرار مکرر این مشکل، شاخصه دسترس پذیری از بین می رود. به جهت جلوگیری قطعی از بروز چنین مشکل هایی، گره پس از دریافت $Token$ جدید ملزم به ارسال پاسخ با استفاده از فرآیند $Exchange$ می باشد. در صورت دریافت پاسخ، سرور $Token$ های قبلی را منقضی کرده و $Token$ جدید را در ارتباطات آتی استفاده می کند. این فرآیند به نوعی از سرگیری جلسه [37] می باشد به طوری که تا حدودی ونه به طور قطع از حمله سرقت جلسه[38] جلوگیری می نماید.

$Chain\ Signature$: مکانیسم شبه زنجیره بلوکی[39] مورد استفاده در پروتکل پیشنهادی که می توان آن را زنجیره امضا درخواست ها[40] نیز نامید، علاوه بر ایجاد صحت بیشتر بر اساس توالی منظمی از درخواست ها می تواند از سرقت و جعل جلسه به طور جدی جلوگیری کند. کارکرد این مکانیسم به گونه ایی است که اگر زنجیره امضا درخواست دچار نقص یا انقطاع شود سرور ابری، کل جلسه و $Token$ را ابطال کرده و گره را مجبور به احراز هویت مجدد می نماید. در اینجا $Chain\ Signature$ هر درخواست که توسط تابع $Chain$ (رابطه 7) ایجاد می گردد در اصل امضایی از امضا درخواست قبلی ترکیب شده با $S_s$ همین درخواست است:

$$Chain = Sha1(Signature_{n-1} + S_s) \quad (7)$$

که در آن $Signature_{n-1}$ امضا درخواست قبلی یعنی درخواست $(n-1)$ ام می باشد. دلیل این ترکیب و امضا نهایی زنجیره، منحصر به فرد کردن توالی زنجیره نسبت به همین درخواست است به گونه ایی که امکان جعل یا تقلب را غیر ممکن می سازد. برای این منظور امضا درخواست قبلی یا درخواست $(n-1)$ ام بایستی در حافظه گره ذخیره سازی گردد. این

---

Session Resumption[37]
Session Hijacking[38]
BlockChain[39]
Verify Signature[40]

مکانیسم بدین صورت از جعل و سرقت جلسه جلوگیری می کند به طوری که اگر جلسه ایی سرقت گردد و هردو همزمان فعال یا در حال تبادل درخواست باشند به طور یقین حداقل یکی از دو اتصال با یک جلسه یکسان، نمی تواند زنجیره امضا صحیحی را به سرور ابری ارایه کند. بنابراین سرور این حمله را سریع شناسایی کرده و آن جلسه را به طور کامل باطل می کند به طوری که گره بایستی مجددا احرازهویت را از سر بگیرد. با این اقدام دسترسی هکر به اطلاعات به وسیله آن جلسه سرقت شده منع و غیر ممکن می گردد. بدین ترتیب تا حد زیادی از حمله سرقت جلسه پیشگیری می شود اما چنانچه هکر دسترسی کامل از راه دور یا فیزیکی پیوسته داشته باشد هیچ پروتکلی به طور منطقی قادر به جلوگیری از این نوع حمله نیست.

همان طور که ذکر شد در توابع $Chain$ و $Tokenize$ از $SH1$ استفاده می شود. بدیهی است که این الگوریتم منسوخ شده اما نیازهای این توابع را برآورده می کند و نیازی به استفاده از درهم سازهای قوی تر نیست. دلیل این امر آن است که دراین توابع دو پارامتر $Token$ و $S_s$ دو مقدار متغیر نسبت به هر تراکنش داده ایی (درخواست) می باشند که ترکیب آن ها در امضا استفاده می گردد و امضا را نسبت به حملات $Brute\ Force$ و $Dictionary$ مقاوم می کند.

در نهایت، پکیج تشکیل شده به شکل شی سریال شده[41] یا $JSON$ می باشد که در نهایت توسط تابع $S_{enc}$ (رابطه 8) رمز نگاری می گردد:

$$S_{enc} = AES_{CBC}\ (Input,\ IV_s, KEY_s) \qquad (8)$$

همان طوری که قبلا توضیح داده شد مقادیر $KEY_S$ و $IV_S$ به طور ثابت در کلیه گره ها و سرور ابری وجود دارد و در رمزنگاری استفاده می گردد. علت استفاده از این رمزنگاری امنیت داده ها در سطح بالا نیست بلکه علاوه بر پنهان نگاری، مدل الگویی الگوریتم در انتقال داده ها در شبکه فاش نمی گردد و رمزگشایی این مکانیسم نیازمند مهندسی معکوس نرم افزار گره ها و همچنین صرف انرژی و پردازش در زمان تبادل داده ها است که این امر از تشخیص الگو پروتکل در مسیریاب ها و ارایه دهنده خدمات اینترنت[42] جلوگیری می نماید.

## 4.4 ارتباطات گره ها با سرور ابری – قسمت پاسخ

پکیج استفاده شده درپاسخ سرور ابری به درخواست ارسالی از گره ها در این قسمت معرفی شده است. همان طوری که در شکل 1.2 نشان داده شده است، این پکیج شامل هفت فیلد می باشد که تمامی این فیلدها به تفصیل توضیح داده شده اند.

| $Signature$ | $Data$ | $DeviceID\ Verification$ | $Extra$ | $Seq\ \#$ | $Token$ | $Time_{Res}$ |
|---|---|---|---|---|---|---|

شکل 2-4 پکیج استفاده شده در ارتباطات گره با سرور ابری- قسمت پاسخ

[41] Serialize
[42] Internet Service Provider

$Signature$: همانند فیلد همنام خود در قسمت درخواست، این فیلد دربردارنده امضا منحصر به فرد هر پاسخ است که نتیجه تابع درهمساز $S_{Sig}$ می باشد و به جهت حفظ یکپارچگی اطلاعات مبادله شده بین سرور ابری و گره به کار می رود. استفاده از فیلد امضا باعث می گردد پاسخ منحصر به درخواست شده و از امکان وقوع جعل جلوگیری می کند. تابع $S_{Sig}$ طبق رابطه زیر (9) تعریف می گردد:

$$S_{Sig} = Hash(S_s + Date\ ('Y') + Request + Data + DeviceID + Token + Seq\# + \text{Extra} + Time_{Res}\ , S_s) \quad (9)$$

این رابطه همانند رابطه (2) می باشد با این تفاوت که پارامترهای دیگری نیز به آن اضافه شده اند که در ادامه توضیح داده خواهد شد.

$Data$: همانند پکیج درخواست، این فیلد شامل اطلاعات خام (رمزنگاری نشده) است که می تواند به فرمت JSON یا Base64 می باشد که با استفاده از تابع رمزنگاری $R_{enc}$ که در رابطه 1 توضیح داده شد رمزنگاری می گردد.

$DeviceID\ Verification$: در پاسخ سرور ابری به گره درخواست دهنده، $DeviceID$ دریافتی با استفاده از تابع $D_{Sig}$ (رابطه 10) امضا می گردد:

$$D_{Sig} = Hash\ (DeviceID + S_S, Token) \quad (10)$$

به منظور عدم حدس زدن، جعل و به خطر افتادن حریم شخصی دستگاه یا گره درخواست دهنده، امضایی از ترکیب $S_S$ و $DeviceID$ با $Token$ فعلی ایجاد و در پکیج پاسخ قرار می گیرد.این کار باعث می گردد در هر پاسخ، امضا منحصربه فرد و متفاوتی از $DeviceID$ ایجاد گردد.

$Extra$: همانند فیلد Data، این فیلد می تواند شامل اطلاعات اضافی باشد که به تشخیص برنامه نویس می تواند اضافه گردد مانند آدرس IP و مقدار تاخیر[43] تبادل اطلاعات با سرور ابری و یا مقدار این فیلد می تواند خالی باشد. برای رمزنگاری اطلاعات موجود در این فیلد از تابع $R_{enc}$ مطابق با رابطه (11) استفاده می گردد:

$$R_{enc} = AES_{CBC}\ (Input, IV_S, Key_R) \quad (11)$$

لازم به ذکر است عملیات فشرده سازی برای هر دو فیلد $Data$ و $Extra$ می تواند قبل از رمزنگاری صورت پذیرد.

$Token$: همانطور که در قسمت قبل توضیح داده شد، $Token$ یک مقدار بسیار تصادفی است که در صورت تشخیص سرور می تواند به صورت مقطعی یا بازه ایی تغییر کند. گره در صورت دریافت $Token$ تغییر پیدا کرده، بایستی با استفاده از تابع $Exchange$ اعلان بدارد که مقدار $Token$ جدید را دریافت کرده است. برای این اعلان محدودیت تعریف شده ایی توسط برنامه نویس وجود دارد و در صورت عدم اعلان سرور ابری، جلسه گره مورد نظر را باطل می کند. چنانچه $Token$

[43] Latency

جدیدی برای تنظیم شدن وجود نداشته باشد، مقدار آن می تواند برابر با $Token$ قبلی یا خالی باشد. مقدار $Token$ با تابع رمزنگاری $R_{enc}$ می گردد.

$Seq\#$: این فیلد شامل همان شناسه عددی است که در در بخش 1.1 در پکیج درخواست قرار داده شد تا چنانچه تعدادی از درخواست ها به صورت موازی ارسال گشتند توالی پاسخ های دریافت شده معین گردد. در صورت عدم وجود پردازش موازی مقدار این فیلد خالی در نظر گرفته می شود.

$Time_{Res}$: این فیلد نشاندهنده زمان پاسخ به درخواست گره توسط سرور ابری می باشد و با استفاده از آن می توان از وقوع حمله MitM جلوگیری کرد.

دو فیلد $Seq\#$ و $Time_{Res}$ نیازی به رمزنگاری ندارند زیرا که هردو در امضا پاسخ شرکت دارند و غیرقابل جعل هستند و چنانچه مورد تغییر یا ویرایش قرار گیرند، امضا غیر معتبر می گردد.

## 4.5 ارتباطات گره ها با سرور ابری – مکانیسم سیگنال

این مکانیسم برای بهینه سازی ارتباطات بین گره و سرور و پشتیبانی از ویژگی بلادرنگی ارائه شده است. ویژگی مهم سیستم سیگنال این است که اتصال بسیار کم حجم و سبکی با سرور دارد و در صورت نیاز به آپدیت، سرور ابری توسط پرچم اعلان آپدیت کامل در قسمت پاسخ سیستم سیگنال، مقدار مثبت عددی را برمی گرداند که معرف وجود نیاز به دریافت آپدیت از سرور ابری مربوطه (اتصال کامل) در بخش تعریف شده نسبت به آن عدد (قابل تعریف توسط برنامه نویس جهت مشخص کردن بخش نیازمند آپدیت) است. چنانچه پرچم عدد صفر را نشان دهد نمایانگر عدم نیاز به اتصال کامل به سرور ابری و دریافت آپدیت در آن زمان است. ارائه عدد منفی بیانگر وجود خطایی در سیستم است که با توجه به عدد بازگشتی نوع خطا مشخص می گردد. بنابراین پاسخ درخواست سیگنال یکی از سه حالت عددی مثبت، منفی یا صفر است و هیچ رمزنگاری هم در این قسمت صورت نمی گیرد. با توجه به عدم رمزنگاری و کوچک بودن طول عدد صحیح نشان دهنده وضعیت آپدیت (معمولا یک رقمی است) سربار پردازشی آن در کمترین حالت ممکن خواهد بود.

در بخش درخواست سیستم سیگنال که توسط گره به سرور ابری فرستاده می شود از رمزنگاری متقارن استفاده شده و پکیج درخواست (شکل 1.3) امضا می گردد. از آن جایی که این پکیج کوچکتر از پکیج درخواست کامل است (در بخش 1.1 توضیح داده شد) و از رمزنگاری نامتقارن RSA (که سربار و مقدار داده ایی بیشتری دارد) در آن استفاده نشده، با وجود حفظ امنیت و محرمانگی سربار پردازشی به حداقل ممکن رسیده است. فیلدهای پکیج درخواست در ادامه توضیح داده شده است:

| $Identifier$ | Data | $Time$ | $Signature$ |
|---|---|---|---|

شکل 3-4 پکیج استفاده شده در ارتباطات گره با سرور ابری- قسمت درخواست سیستم سیگنال

$Identifier$: سیستم سیگنال به منظور مشخص شدن هویت دستگاه (یا جلسه) و یافتن اطلاعات در سرور ابری به یک شناسه (یا جلسه) از دستگاهی که قبلا در سرور (در هنگام Auth) ثبت شده نیاز دارد. این مقدار در هنگام ثبت جلسه یا $DeviceID$ و احراز هویت توسط سرور ایجاد گشته و به دستگاه اعلان می گردد. فیلد $Identifier$ یا شناسه دستگاه یا جلسه (بسته به نوع احراز هویت) ترکیبی از مقدار $DeviceID$ و امضا آخرین پکیج ارسالی و ترکیب نهایی با $Token$ می باشد تا از جعل، دستکاری یا کشف آن تا حد زیادی جلوگیری شود و با استفاده از تابع $N_{Sig}$ محاسبه می گردد:

$$N_{Sig} = Hash\,(DeviceID\,, Token) \tag{12}$$

$Data$: این فیلد می تواند شامل اطلاعات مختصری برای تعیین وجود و نوع آپدیت به دو فرمت JSON یا Base64 باشد که با استفاده از تابع رمزنگاری $N_{enc}$ مطابق رابطه 11 رمزنگاری می گردد:

$$N_{enc} = AES_{CBC}(Input, IV_S, KEY_{RT}) \tag{13}$$

که در آن $KEY_{RT}$ کلید چرخشی متغیر قابل استفاده در $AES_{CBC}$ می باشد که از ترکیب کلید ثابت $KEY_S$ و $Token$ حاصل می گردد. پیشنهاد می گردد فیلد $Data$ در مکانیسم سیگنال حدالامکان حاوی اطلاعات حیاتی یا هویتی نباشد. زیرا با وجود این که رمزنگاری کاملی در این مرحله صورت می گیرد به دلیل عدم چرخش دائمی کلید رمزنگاری برای هر پیام به صورت مجزا (برخلاف مراحل دیگر) داده در معرض ریسک افشای اطلاعات می باشد. همچنین تابع $N_{enc}$ داده های ورودی اش را با توجه به کلید چرخشی ایجاد شده توسط $Token$ و امضا آخرین درخواست رمزنگاری می کند. بنابراین عدم تغییر $Token$ یا امضا آخرین درخواست در مدت طولانی می تواند ریسک بیشتری برای این رمزنگاری به همراه داشته باشد.

$Time$:همانند فیلد همنام خود در بخش1.1، این فیلد نشاندهنده زمان حال حاضر درخواست گره از سرور ابری می باشد.

$Signature$: همچون دیگر قسمت ها، امضا برای اطمینان از صحت داده ها و جلوگیری از جعل آن ها در پکیج سیگنال قرار داده شده است که ترکیبی از فیلدهای $DeviceID$، $Data$ و $Time$ با $Token$ می باشد و توسط تابع $M_{Sig}$ مطابق با رابطه (14) ایجاد می گردد:

$$M_{Sig} = Hash(DeviceID + Data + Time\,, Token) \tag{14}$$

در نهایت کل پکیج توسط تابع $S_{enc}$ (رابطه 8) رمزنگاری می گردد تا الگوی آن همان طوری که در قسمت های قبل توضیح داده شد، قابل شناسایی نباشد. مکانیسم سیگنال برای ارتباطات امن و بلادرنگ یک راهکار کامل به شمار می آید که سربار پردازشی و در نتیجه انرژی و مصرف پهنای باند شبکه را به طور چشمگیری کاهش می دهد. در واقع در صورت نیاز به آپدیت کامل، درخواست در اولین زمان ممکن به طور سریع و به اصطلاح بلادرنگ توسط گره درخواست کننده ارسال و داده های آپدیت در قالب پاسخ دریافت می گردد.

## 4.6 ارتباطات غیرمتمرکزبین گره ایی – قسمت درخواست

پکیج استفاده شده در یک درخواست ارسالی از یک گره به گره دیگر در این قسمت معرفی شده است. همان طوری که در شکل 1.2 نشان داده شده است، این پکیج شامل هشت فیلد می باشد که تمامی این فیلدها به تفصیل توضیح داده شده اند.

| $Request$ | $Data$ | $Label$ | $Signature$ | $Communication\ Salt$ | $PeerID$ | $Time_{Req}/Salt$ | $Seq\#$ | $MyRoute$ |
|---|---|---|---|---|---|---|---|---|

شکل 4-4 پکیج استفاده شده در ارتباطات گره با گره- قسمت درخواست

$Request$: این فیلد دربردارنده رشته متنی است که مشخص می کند چه منبع یا روال از گره دیگری مورد درخواست می باشد. این فیلد احتیاجی به رمزنگاری ندارد.

Data: این فیلد شامل اطلاعات خام (رمزنگاری نشده) است که می تواند به فرمت JSON یا Base64 می باشد که با استفاده از تابع رمزنگاری $OR_{enc}$ مطابق رابطه (15) رمزنگاری می گردد:

$$OR_{enc} = AES_{CBC}(Input, IV_S, KEY_{R_P}) \quad (15)$$

که در آن $KEY_{R_P}$ کلید چرخشی متغیر قابل استفاده در $AES_{CBC}$ می باشد که از ترکیب کلید ثابت $KEY_S$ و مقدار تصادفی $S_c$ حاصل می گردد که در ادامه توضیح داده خواهد شد.

$Label$: یک شناسه منحصر به فرد و مختص هر گره است که به مسیریابی بین گره ایی کمک می کند.

$Signature$ : این فیلد دربردارنده امضا منحصر به فرد هر درخواست است که نتیجه تابع درهمساز $A_{Sig}$ می باشد و به جهت حفظ یکپارچگی اطلاعات مبادله شده بین گره ها به کار می رود. تابع $A_{Sig}$ طبق رابطه زیر تعریف می گردد:

$$A_{Sig} = Hash(S_c + Date\ ('Y') + Request + Data + PeerID + Seq\# + Time_{Req} + MyRoute\ , S_c) \quad (16)$$

امضا پکیج درخواست در ارتباطات بین گره ایی همانند امضا پکیج در ارتباطات گره ها با سرور ابری نقش بسیار مهمی را ایفا می کند و از جعل، تغییر و آنالیز بسته توسط حمله کنندگان در حمله MitM جلوگیری می کند. ترکیبی بودن این امضا آن را از این جهت متمایز می کند که با وجود قسمت های رمزنگاری نشده در پکیج درخواست، امکان کوچک ترین تغییری در آن وجود ندارد.

$Communication\ Salt$: برای تولید این فیلد، ابتدا گره یک مقدار بسیار تصادفی را ایجاد می کند که آن را $S_c$ می نامیم. سپس، $S_c$ با استفاده از الگوریتم رمزنگاری RSA و کلید عمومی گره مقصد $(Peer\ PKEY)$ مطابق رابطه (17) رمزنگاری می گردد.

$$RSA_c = RSA_{PK_P}\ (Input) \quad (17)$$

بایستی توجه داشت که همانند ارتباطات گره ها با سرور ابری (بخش 1.1)، $S_c$ در رمزنگاری دیگر بخش ها حضور دارد و کلید چرخشی رمزنگاری، $KEY_{R_P}$ را تولید می کند با این تفاوت که تابع رمزنگاری $RSA_c$ ، مقدار $S_c$ را با کلید عمومی گره مقصد که پذیرنده درخواست است برای رمزنگاری استفاده می کند و کلید عمومی سرور ابری در این قسمت استفاده نمی گردد. $Communication\ Salt$ مقدار رمزنگاری شده $S_c$ می باشد که با توجه به جدول Peer Table و با استفاده از کلید عمومی گره مقصد درخواست شونده بدست می آید. لازم به ذکر است این جدول برای هر یک از گره ها از حیث $MyID$ و $PeerID$ منحصر به فرد می باشد و هر گره موظف است اطلاعات زیر را از خود ایجاد و به سرور اعلان کند.

جدول 2-4 یک نمونه ازجدول Peer Table

| $Label$ | $Endpoint$ | $MyID$ | $PeerID$ | $Peer\ PKey$ |
|---|---|---|---|---|
| $NR1$ | 88… | 70 | 44 | P1 |
| $SMP3$ | 216… | 44 | 26 | P2 |
| $EXP1$ | 96… | 32 | 46 | P3 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

$PeerID$: در ارتباطات بین گره ایی، فیلد PeerID جایگزین فیلد DeviceID در ارتباطات گره با سرور ابری می گردد که عملکردی مشابه DeviceID دارد ولی به روش متفاوتی ایجاد می گردد و بستگی به جدول Peer Table که گره دریافت می کند دارد. همان طوری که در جدول 1-2 نشان داده شده است مقدار MyID، متناظر با PeerID که گیرنده پیام درخواست است قرار می گیرد و توسط تابع $OR_{enc}$ رمزنگاری می گردد و عینا در گره دریافت کننده نیز با توجه به جدول آن گره تایید صحت و برای پاسخ این مقدار آماده سازی می گردد.

$Time_{Req}/Salt$: همانند ارتباطات گره با سرور ابری، این فیلد می تواند نشاندهنده زمان حال حاضردرخواست گره ($Time_{Req}$) یا شمارنده کل درخواست های ارسالی ($Salt$) باشد. در معماری پیشنهادی از زمان حال حاضر استفاده گشته ولی معماری های دیگر می توانند مستقل از زمان و بر اساس پارامتر دیگری همچون شمارنده کل درخواست ها طراحی گردند.

$\#Seq$: این فیلد همانند فیلد همنام خود در ارتباطات گره با سرور ابری، یک شناسه عددی است که در پکیج درخواست قرار داده می شود تا اگر تعدادی از درخواست ها به صورت موازی ارسال شوند توالی پاسخ های دریافت شده معین گردد.

$MyRoute$: جهت تولید این فیلد، ابتدا یک شی با نام $RO$ از نوع آرایه یا مجموعه ایی متشکل از چهار عنصر Endpoint ، Public Key ، MyID و PeerID جدید آن گره درخواست کننده مطابق رابطه (18) ایجاد می گردد:

$$RO = Array(Endpoint, Pkey, PeerID, MyID, Label) \quad (18)$$

سپس $RO$ توسط تابع $OR_{enc}$ رمزنگاری می گردد و فیلد MyRoute حاصل می شود. MyRoute در اصل به گره مقصد اعلام می دارد تا برای پاسخ از جدول و مشخصه های اعلام شده جدید به جای مقادیر قبلی آن استفاده کند. این مکانیسم برای طراحی معماری های امنیتی بسیار قوی و سطح بالا و بعضا با مصارف نظامی طراحی و تعبیه شده است تا علاوه بر امکان به روزرسانی بلادرنگ گره های تعاملی، بتواند با ایجاد مسیرهای جدید و کلیدهای رمزنگاری، سطح بالایی از محرمانگی را فراهم نماید. ترجیحا پیشنهاد می گردد تا تغییرات این چنینی سریعا به سرور ابری نیز اعلان گردد تا گره های دیگر مسیریابی خود را بدون نقص و مشکل انجام دهند.

همان طوری که مشاهده کردید در ارتباطات میان گره ایی، پکیج داده ها مشابه ارتباطات گره ها با سرور ابری می باشد اما تفاوت هایی نیز دارد که در ادامه به آن می پردازیم:

به دلیل محدودیت های محاسباتی دستگاه ها و گره ها در این حالت، فیلدهای $Token$ و $Chain\ Signature$ وجود ندارند و عملا به دلیل همگام سازی ها و کنترل کننده های سرور، در ارتباطات بین گره ها نیازی به این مکانیسم ها نمی باشد. همچنین از آن جایی که گره ها برای دریاف جدول $Peer\ Table$ توسط سرور یک بار بررسی و تایید هویت می گردند، لذا جعل این جدول ناممکن بوده و با وجود مکانیسم کلید چرخشی دوطرفه، نیازی به این دو مکانیسم نمی باشد. از طرفی به دلیل محدود منابع محاسباتی و انرژی در گره ها ایجاد و ذخیره جداول Hash از ارتباطات و کنترل کننده ها به وسعت سرور امکان پذیر نیست و این جداول باید به صورت بسیار بهینه ایی مورد استفاده قرار گیرد.

در نهایت، کل پکیج رمزنگاری شده و ارسال می گردد. این رمزنگاری انتهایی جنبه امنیت به تنهایی نداشته و به منظور محرمانگی و جلوگیری از تشخیص الگوی درخواست همراه با امنیت نسبی و لایه نهایی ارایه گردیده است. این مکانیسم توسط تابع $S_{enc}$ ارایه می گردد. لازم به توضیح است که مکانیسم های مسیریابی شبکه مستقل از مکانیسم پروتکل پیشنهادی است وانتخاب آن به طراح سیستم واگذار می گردد. هدف پروتکل پیشنهادی صرفا امن سازی بستر ارتباطی می باشد.

## 4.7 ارتباطات غیرمتمرکز بین گره ایی – قسمت پاسخ

پکیج استفاده شده در پاسخ به درخواست ارسالی از یک گره به گره دیگر در این قسمت معرفی شده است. همان طوری که در شکل 1.4 نشان داده شده است، این پکیج شامل هفت فیلد می باشد که تمامی این فیلدها در ادامه به تفصیل توضیح داده شده اند.

| *Signature* | *Data* | *Label* | *Sodium* | *PeerID Verification* | *Seq #* | *Time/Salt* | *MyRoute* |
|---|---|---|---|---|---|---|---|

شکل 5-4 پکیج استفاده شده در ارتباطات گره با گره- قسمت پاسخ

$Signature$: این فیلد دربردارنده امضا منحصر به فرد هر پاسخ است که نتیجه تابع درهمساز $B_{Sig}$ می باشد و همچون امضا درخواست به جهت حفظ یکپارچگی اطلاعات مبادله شده بین گره ها به کار می رود. فیلد امضا نقش مهم تایید صحت داده ها را تضمین و از امکان وقوع جعل جلوگیری می کند. تابع $B_{Sig}$ طبق رابطه زیر (19) تعریف می گردد:

$$B_{Sig} = Hash(S_c + Date\ ('Y') + Request + Data + PeerID + S_p + Seq\# + \text{Time} + \text{MyRoute}, S_c) \quad (19)$$

این رابطه همانند رابطه (16) می باشد با این تفاوت که پارامتر $S_p$ به آن اضافه شده است که در ادامه توضیح داده خواهد شد.

$Data$: این فیلد شامل اطلاعات خام (رمزنگاری نشده) است که می تواند به فرمت JSON یا Base64 می باشد که با استفاده از تابع رمزنگاری $TR_{enc}$ (رابطه 20) رمزنگاری می گردد:

$$TR_{enc} = AES_{CBC}(Input, IV_S, KEY_{R_T}) \quad (20)$$

که در آن $KEY_{R_T}$ کلید چرخشی متغیر قابل استفاده در $AES_{CBC}$ می باشد که از ترکیب کلید ثابت $KEY_S$ و مقدارهای تصادفی $S_c$ و $S_p$ حاصل می گردد که در ادامه توضیح داده خواهد شد.

$Sodium$: در این قسمت، مواردی همچون $Token$ و زنجیره درخواست ها وجود ندارند ولی در طراحی معماری ارتباطی میان گره ایی فیلدی به نام $Sodium$ تعبیه شده است که نقش آن ایجاد یک بستر دوطرفه با امنیت بالا می باشد. به دلیل وجود معماری خاص و ویژه ارتباطات میان گره ایی، کلیه گره های مشارکت داده شده، کلیدهای عمومی یکدیگر را دارند و این قابلیت امکان پیاده سازی ساختاری جهت ایجاد $Sodium$ را فراهم می نماید.

برای تولید این فیلد، ابتدا گره یک مقدار بسیار تصادفی را ایجاد می کند که آن را $S_p$ می نامیم. $S_p$ یک عنصر ویژه جهت ایجاد کلید چرخشی است. در حقیقت، $S_p$ و $Sodium$ با $S_c$ و $Communication\ Salt$ وظیفه مشابهی دارند ولی در امضا و رمزنگاری پکیج پاسخ به گره درخواست دهنده استفاده می شوند. $S_p$ با استفاده از الگوریتم رمزنگاری RSA و کلید عمومی گره مقصد $(Peer\ PKEY)$ مطابق رابطه (21) رمزنگاری می گردد.

$$RSA_p = RSA_{PK_T}(Input) \quad (21)$$

بایستی توجه داشت که، ترکیب $S_p$ و $S_c$ در رمزنگاری دیگر بخش ها حضور دارد و کلید چرخشی رمزنگاری، $KEY_{R_T}$ را تولید می کنند با این تفاوت که تابع رمزنگاری $RSA_p$ ، مقدار $S_p$ را با کلید عمومی گره مقصد که فرستنده درخواست است برای رمزنگاری استفاده می کند. $Sodium$ مقدار رمزنگاری شده $S_p$ می باشد که با توجه به جدول Peer Table و با استفاده از کلید عمومی گره مقصد درخواست کننده بدست می آید. لازم به ذکر است این جدول برای هر یک از گره ها از حیث $MyID$ و $PeerID$ منحصر به فرد می باشد.

$PeerIDVerification$: وظیفه این فیلد تایید صحت پاسخ به گره درخواست دهنده می باشد که در پاسخ به آن، $PeerID$ دریافتی با استفاده از تابع $D_{Sig}$ (رابطه 22) امضا می گردد:

$$D_{Sig} = Hash\ (PeerID + S_c, S_p) \qquad (22)$$

این کار باعث می گردد در هر پاسخ، امضا منحصربه فرد و متفاوتی از $PeerID$ ایجاد گردد.

سه فیلد $Seq\#$ ، $Time/Salt$ و $Label$ همانند سه فیلد همنام خود در قسمت درخواست ارتباطات بین گره ایی تعریف می گردند.

$MyRoute$: همانند فیلد همنام خود در قسمت درخواست ارتباطات بین گره ایی، ابتدا یک شی با نام $RO$ از نوع آرایه متشکل از چهار عنصر Endpoint ، Public Key ، MyID و PeerID جدید آن گره درخواست کننده مطابق رابطه 18 ایجاد می گردد. سپس $RO$ توسط تابع $TR_{enc}$ رمزنگاری می گردد و فیلد MyRoute حاصل می شود. MyRoute در صورت نیاز به گره درخواست کننده اعلام می دارد تا از جدول و مشخصه های اعلام شده جدید (شامل کلیدهای جدید) برای ارتباطات بعدی استفاده بکند.

این مکانیسم برای طراحی معماری های امنیتی بسیار قوی و سطح بالا و بعضا با مصارف نظامی طراحی و تعبیه شده است تا علاوه بر امکان به روزرسانی بلادرنگ گره های تعاملی، بتواند با ایجاد مسیرهای جدید و کلیدهای رمزنگاری، سطح بالایی از محرمانگی را فراهم نماید. ترجیحا پیشنهاد می گردد تا تغییرات این چنینی سریعا به سرور ابری نیز اعلان گردد تا گره های دیگر مسیریابی خود را بدون نقص و مشکل انجام دهند.

# 5 پیشنهاد های پیاده سازی

در اینجا پیشنهاد می گردد با وجود انعطاف پذیر بودن این پروتکل، برای اجرای کامل آن، حداقل محدودیت های زیر اعمال گردد:

## 5.1 رمزنگاری متقارن

- بایستی از الگوریتم $AES$ [44] در حالت های $CBC$ [45] و $IGE$ [46] استفاده گردد.
- به دلیل وجود کلید چرخشی در رمزنگاری پکیج ها، طول کلید مناسب برابر با 128 بیت یا 16 کاراکتر پیشنهاد می گردد که این 16 کاراکتر می تواند شامل کلیه مقادیر جدول اسکی باشد. این طول کلید سربار پردازشی کمتر نسبت به مقادیر بیشتر دارد.

---

[44] Advanced Encryption Standard
[45] Cipher Block Chaining
[46] Infinite Garble Extension

- ازرمزنگاری بدون $Padding$ داده ها اجتناب و $Padding$ مناسب با توجه به حالت استفاده شده الگوریتم انتخاب گردد.
- در صورت استفاده از حالت $CBC$ حتما از $PKCS7$ [47] برای $Padding$ استفاده شود.
- برای مصارف نظامی و نیازمند امنیت بسیار بالا، از حالت پیشرفته $GCM$ [48] استفاده گردد.

## 5.2 رمزنگاری نامتقارن

- از الگوریتم $RSA$ [49] با طول کلید 2048 یا 4086 استفاده گردد.
- حتما از $OAEP$ [50] برای $Padding$ استفاده گردد.
- کلید خصوصی سرور ابری به طور کامل و بی نقصی محافظت گردد.

## 5.3 درهم سازی $(Hashing)$

- برای درهم سازی داده ها و امضا آن ها، تابع $Hash$ یا مشابه آن که از ترکیب $Salt$ و درهم سازهای $SHA256$ یا بالاتر استفاده می کند پیشنهاد می گردد.
- تابع دو جمله ایی (دو حالته) $Hash$ به منظور ایجاد احتمال کمتر در بوجود آمدن $Collision$ به این شکل ایجاد شده است و جنبه دیگری ندارد.
- تابع $Hash$ می تواند بنا بر تمایل طراح سیستم به گونه های دیگری نیز طراحی شده و به کار گرفته شود ولی می بایست بین سرور و گره یکسان باشد.

## 5.4 مقادیر بسیار تصادفی $(Highly\ Random)$

- برای تولید مقدار های عددی، حروف، یا ترکیب عدد و حروف به صورت تصادفی می توان از توابع $Rand$ ریاضی، زمان، درهم سازها یا ترکیب آن ها برای بهبود عملکرد تولید مقادیر بسیار تصادفی استفاده کرد.
- بنا به تمایل طراح سیستم از هر راهکاری می توان برای تولید مقادیر تصادفی استفاده کرد ولی لازم به ذکر این نکته است که جنبه تصادفی تولید این مقادیر باید بسیار قوی باشد.

---

[47] Public Key Cryptographic Standard
[48] Galois/Counter Mode
[49] *Rivest–Shamir–Adleman*
[50] Optimal Asymmetric Encryption Padding

### 5.5 تولید مقادیر خاص

- یکی از متغیرهای بسیار مهمی که می بایست در هر درخواست ایجاد گردد مقدار $S_S$ است که حداکثر طول آن بایستی با توجه به طول کلیدهای رمزنگاری متقارن تعیین گردد.
- با توجه به طول کلید پیشنهادی 128 بیتی الگوریتم $AES$ ، پیشنهاد می گردد حداقل و حداکثر طول $S_S$ ، به ترتیب 12 و 16 کاراکتر در نظر گرفته شود و ترکیبی از اعداد و حروف باشد تا در برابرحملات $Brute\ Force$ و $Dictionary$ مقاوم گردد.
- مقدار تولید شده بایستی به صورت بسیار تصادفی ایجاد گردد تا در بازه مجاز زمان که توسط سرور ابری بررسی می گردد امکان تکرار دو مقدار $S_S$ به حداقل برسد.
- مقدار متغیر $KEY_R$ به دلیل متغیر بودن طول $S_S$ به صورت ترکیب دو متغیر $KEY_R$ و $S_S$ بوجود می آید.

### 5.6 امضا

- همان طوری که توضیح داده شد، امضا بایستی شامل پارامترهایی باشد که در معماری این پروتکل آورده شده است تا حداقل توان ایمنی را تضمین کند. اما پارامتری مانند $Date\ ('Y')$ به صورت انتخابی است و چنانچه حذف گردد مشکل حادی ایجاد نمی شود.
- همچنین پارامترهای دیگری می توانند به این امضا اضافه گردند اما باید دقت داشت که توسط سرور قابل محاسبه باشند و سپس در تایید صحت پکیج استفاده شوند.
- پارامترهایی که در امضا شرکت داده شده اند به صورت خام یا قبل از رمزنگاری می باشند. کوچک ترین تغییر در آن ها باعث تغییر در امضا خواهد شد، بنابراین امضا غیر قابل جعل می باشد. از طرفی برخی پارامترهای شرکت کننده مانند $Request$ ، $Seq\#$ و $Time$ به دلیل کاهش سربار پردازشی واین که اطلاعات هویتی را فاش نمی کنند رمزنگاری نمی شوند اما به دلیل آن که در امضا شرکت دارند غیرقابل جعل می گردند.

## 6 نمودار های جریان (Flow Diagrams)

در این بخش نمودارهای جریان پروتکل پیشنهادی بر اساس سناریوهای زیر به نمایش درآمده است:

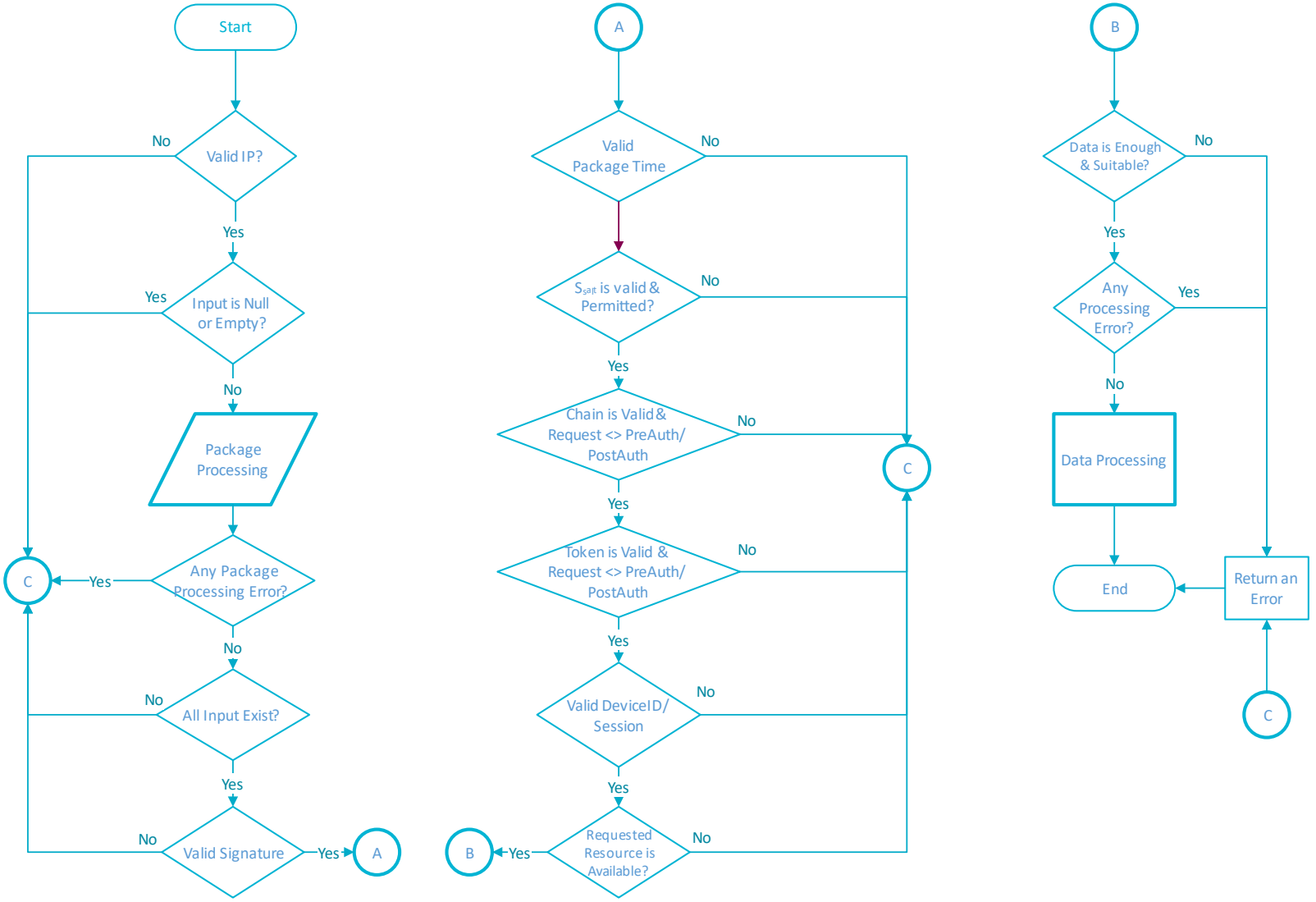
1 ) ارتباطات گره با سرور ابری - درخواست گره از سرور ابری

2 ) ارتباطات گره با سرور ابری - پاسخ سرور ابری به درخواست گره

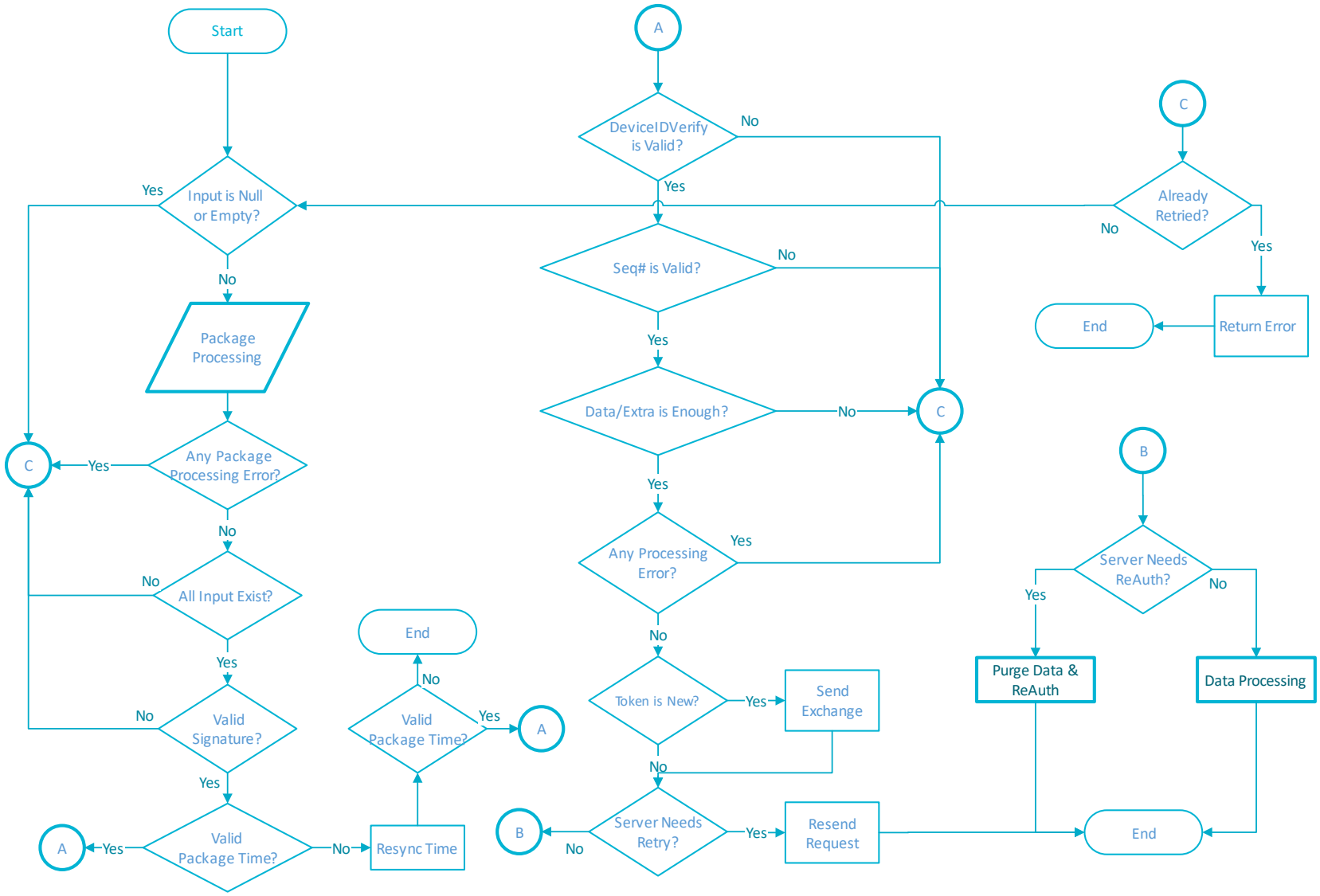
3 ) ارتباطات گره با سرور ابری – سیستم سیگنال

4 ) ارتباطات غیر متمرکز بین گره ایی – درخواست گره از گره

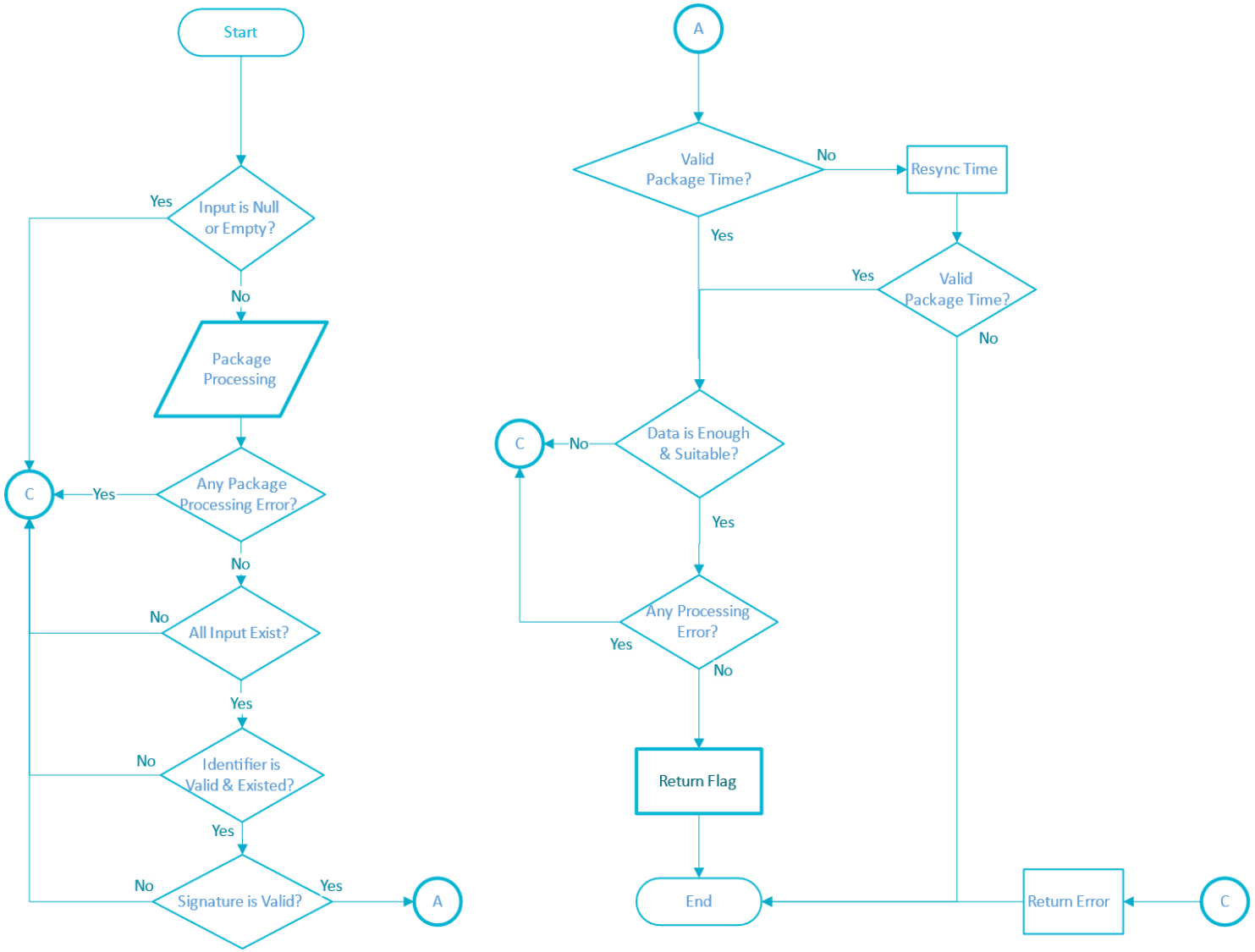
5 ) ارتباطات غیر متمرکز بین گره ایی - پاسخ گره به گره



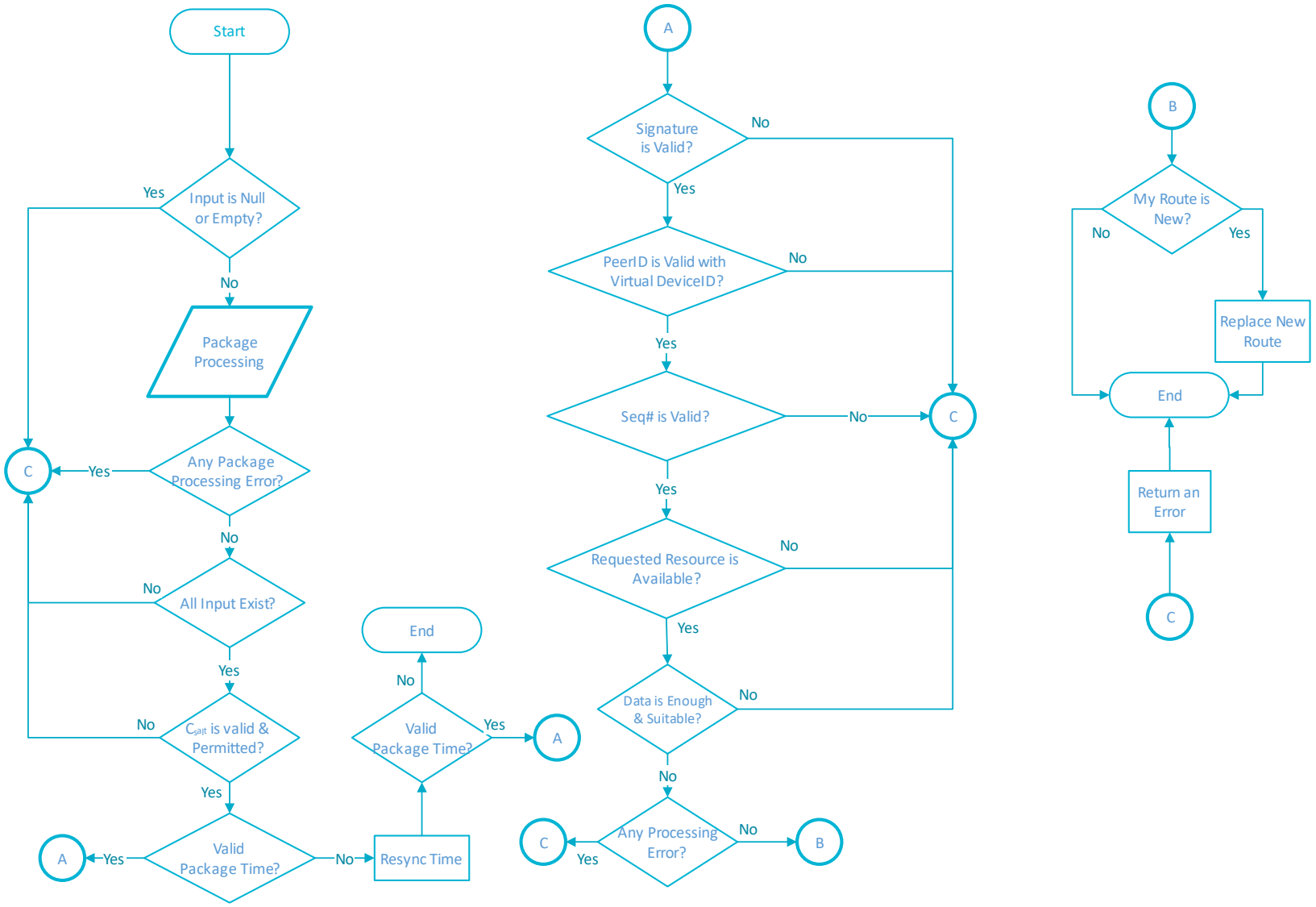

شکل 1-6 ارتباطات گره با سرور ابری - درخواست گره از سرور ابری



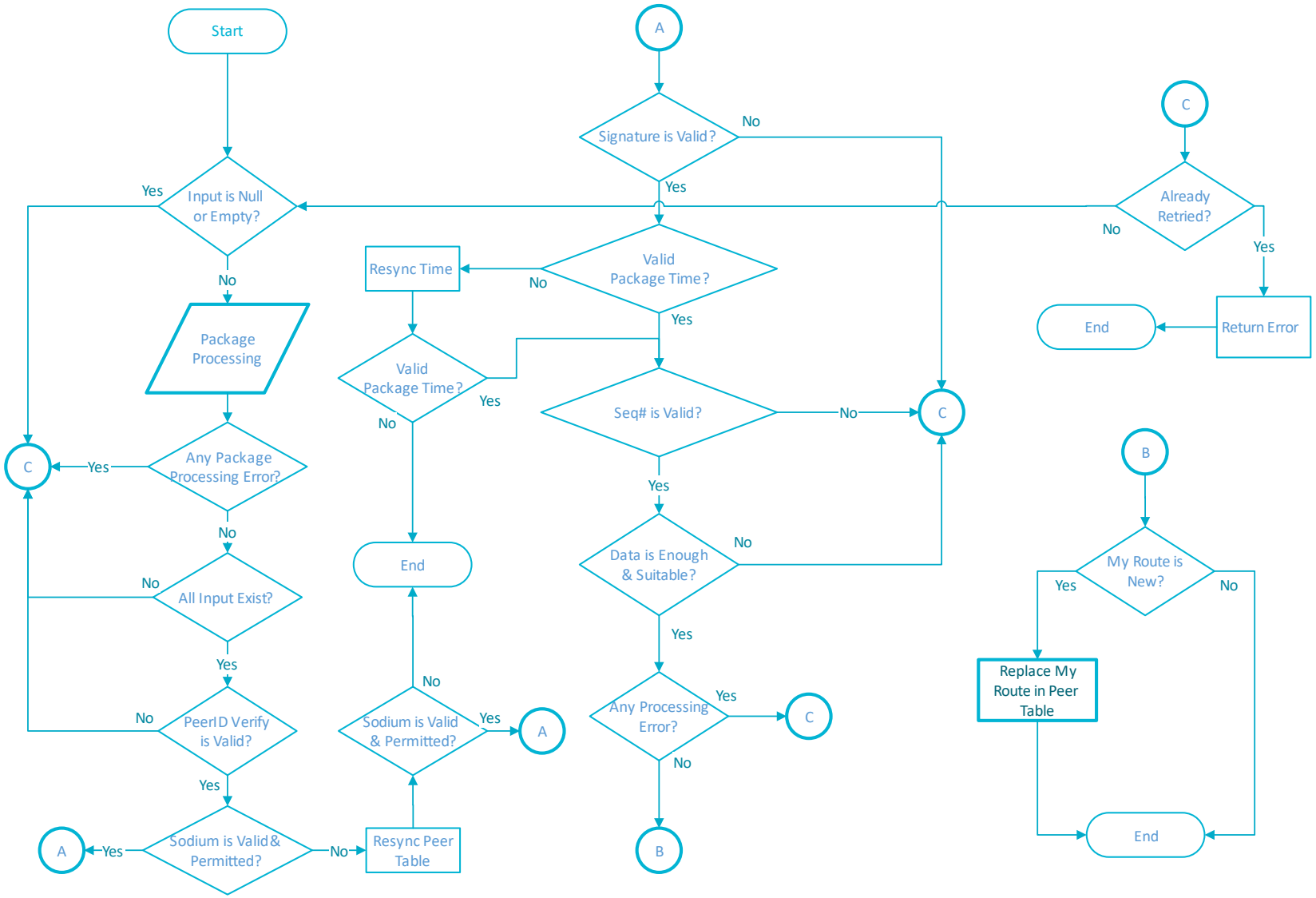

شکل 2-6 ارتباطات گره با سرور ابری - پاسخ سرور ابری به درخواست گره




شکل 3-6 ارتباطات گره با سرور ابری – سیستم سیگنال




شکل 4-6 ارتباطات غیر متمرکز بین گره ایی – درخواست گره از گره




شکل 5-6 ارتباطات غیر متمرکز بین گره ایی - پاسخ گره به گره

## 6.1 پردازش پکیج

همان طوری که در قسمت قبل مشاهده شد، در اولین مرحله تمامی دیاگرام های فوق (بعد از اطمینان از وجود ورودی) پکیج درخواست یا پاسخ پردازش می گردید. در این بخش، جزییات فیلدهایی که در این مرحله پردازش می گردند بر اساس نوع ارتباطات به نمایش در آمده است:

### 6.1.1 درخواست گره از سرور ابری

$$Input = DecryptAES\ (S_{enc}, Input)$$

$$Input = JSONDecode\ (Input)$$

$$Request = Input\ [Request]$$

$$DeviceID = Input\ [DeviceID]$$

$$S_{Salt} = DecryptRSA\ (RSA_P, Input[S_{Salt}])$$

$$Time = Input\ [Time]$$

$$Seq = Input\ [Seq]$$

$$Token = Input\ [Token]$$

$$Signature = Input\ [Signature]$$

$$ChainSignature = Input\ [chain]$$

$$Data = DecryptAES\ (R_{enc}, Input\ [Data], S_{Salt})$$

### 6.1.2 پاسخ سرور ابری به درخواست گره

$$Input = DecryptAES\ (S_{enc}, Input)$$

$$Input = JSONDecode\ (Input)$$

$$DeviceIDVerify = Input\ [DeviceIDVerify]$$

$$Data = DecryptAES\ (R_{enc}, Input[Data])$$

$$Extra = DecryptAES\ (R_{enc}, Input[Extra])$$

$$Seq = Input\ [Seq]$$

$$Time = Input\ [Time]$$

$$Token = DecryptAES(R_{enc}, Input[Token])$$

### 6.1.3 درخواست گره از سرور ابری- سیستم سیگنال

$$Input = DecryptAES\ (S_{enc}, Input)$$

$$Input = JSONDecode(Input)$$

$$IDentifier = Input\ [IDentifier]$$

$$Data = DecryptAES\ (N_{enc}, Input[Data], Token + Signature_{n-1})$$

$$Time = Input\ [Time]$$

$$Signature = Input\ [Signature]$$

6.1.4 درخواست گره از گره (ارتباطات غیر متمرکز)

$$Input = DecryptAES\ (S_{enc}, Input)$$

$$Input = JSONDecode\ (Input)$$

$$Request = Input\ [Request]$$

$$C_{Salt} = DecryptRSA\ (RSA_C, Input[C_{Salt}])$$

$$Data = DecryptAES\ (OR_{enc}, Input[Data], C_{Salt})$$

$$Signature = Input\ [Signature]$$

$$PeerID = DecryptAES\ (OR_{enc}, Input[PeerID], C_{Salt})$$

$$Time = Input\ [Time]$$

$$Seq = Input\ [Seq]$$

$$Route = DecryptAES\ (OR_{enc}, Input[MyRoute], C_{Salt})$$

6.1.5 پاسخ گره به گره (ارتباطات غیر متمرکز)

$$Input = DecryptAES\ (S_{enc}, Input)$$

$$Input = JSONDecode\ (Input)$$

$$Signature = Input\ [Signature]$$

$$Sodium = DecryptRSA\ (RSA_P, Input[Sodium])$$

$$PeerIDVerify = Input[PeerIDVerify]$$

$$Data = DecryptAES(TR_{enc}, Input[Data], Sodium + S_C)$$

$$Seq = Input\ [Seq]$$

$$Time = Input\ [Time]$$

$$MyRoute = DecryptAES\ (TR_{enc}, Input[MyRoute], Sodium + S_C)$$

# 7 شبه کدهای (Pseudocodes) پروتکل پیشنهادی

در این قسمت شبه کدهای پروتکل پیشنهادی بر اساس ارتباطات متمرکز و غیر متمرکز شرح داده شده در قسمت های قبلی به نمایش درآمده است.

## 7.1 شبه کدهای ارتباطات سرور ابری با گره – سمت سرور ابری

1: Array RPack = {};
2: String Input = Null;
3: Input = getInput(.);
4: If (InputMethod is Not Valid OR Input is Null or Empty)
5: {
6: Error (Input is Invalid or Empty);
7: }
8: Try
9: {
10: Input = $S_{dec}(Input, IV, KEY)$;
11: RPack = JSONDecode(Input);
12: If (All input Field are Not exact and Not exist even empty strings)
13: {
14: Error (Input is Invalid or Empty);
15: }
16: String Request = RPack [Request];
17: String Signature = RPack [Signature];
18: String Time = RPack [Time];
19: String Seq = RPack [Seq];
20: String Token = RPack [Token];
21: String Chain = RPack [Chain];
22: String $S_s = RSA_s(RPack[S_{salt}], PrivateKey)$;
23: String Data = $R_{Dec}(RPack[Data], IV, Frag(KEY, S_s))$;
24: String DeviceID = $R_{Dec}(RPack[DeviceID], IV, Frag(KEY, S_s))$;

25: String SigVerify $= C_{Sig}(S_s + Date(Year) + Request + Data + DeviceID + Token + Seq + Time, S_s)$;

26: }

27: Catch

28: {

29: Error (Error on Processing the Request, Maybe Wrong encryption);

30: }

31: If (Signature Not Equals SigVerify)

32: {

33: Error (Wrong Signature);

34: }

35: If (Now+24H<Time<Now-24H)

36: {

37: Error (Request Package Time is invalid, Resync time first!);

38: }

39: If (DeviceID not exist OR invalid OR Ban)

40: {

41: Error (Invalid DeviceID);

42: }

43: String LSignature = getLSignature();

44: LSignature = Tokenize (LSignature, $S_s$);

45: If (Chain Not Equals LSignature And Request Not Equals PreAuth,PostAuth,…)

46: {

47: Error (Chain is invalid, you need to ReAuth);

48: Goto ForceReAuth();

49: }

50: String LToken = getLToken();

51: LToken = Tokenize (LToken, $S_s$);

52: If (Token is NOT Equals LToken AND Request NOT Equals PreAuth,PostAuth,…)

53: {

54: Error (Token is invalid);
55: }
56: Bool $S_s flag = Validate\ S_{salt}(S_s, Time)$;
57: If ($S_S flag\ Equals\ treu$)
58: {
59: setLSignature (signature);
60: record$S_{salt}(S_s, Time)$;
61: }
62: Else
63: {
64: Error (invalid $S_{salt}$, Retry Sending Request);
65: }
66: If (Request Equals 'Exchange')
67: {
68: String CandidateToken = getCandidateToken();
69: CandidateToken = Tokenize (CandidateToken, $S_s$);
70: If (CandidateToken Equals Data)
71: {
72: putCandudateToken As Token();
73: }
74: }
75: If (Request is valid AND exist)
76: {
77: String Res = ProcessMethod(Request);
78: }
79: Else
80: {
81: Error (Request is invalid);
82: }
83: SetIdentifier (DeviceID, NSig (DeviceID+Signature,Token)); //for signal
84: Array RespPack{};

| //Authorization and New Token Generation and others here |
|---|

85: RespPack [Data] = $R_{enc}(Res, IV, Frag(KEY, S_s))$;

86: RespPack [Extra] = $R_{enc}(Extra, IV, Frag(KEY, S_s))$;

87: RespPack [Seq] = Seq;

88: String RTime = now();

89: RespPack [Time] = RTime;

90: String CTok = getCandidateToken();

91: If (CTok is NOT Empty OR Null)

92: {

93: Token = CTok;

94: }

95: else {Token = ' ';}

96: RespPack [Token] = $R_{enc}\big(Token, IV, Frag(KEY, S_s)\big)$;

97: RespPack [DeviceIDVerify] = $D_{Sig}(DeviceID + S_s, Token)$;

98: RespPack [Signature] = $S_{sig}(S_s + Res + Date(year) + Seq + Token + RTime + Request + Extra + DeviceIDVerify, S_s)$;

99: String RespData = JSONEncode (RespPack);

100: RespData = $S_{enc}(RespData, IV, KEY)$;

101: SendResponse (Endpoint,RespData);

### 7.2 شبه کدهای ارتباطات سرور ابری با گره – سمت گره

1: Array Package = {};

2: String Request = "TestResource";

3: Package [Request] = Request;

4: String Data = "MyData";

5: Int Seq = 1;

6: Package [Seq] = Seq;

7: String $S_s = HighRandom()$;

8: Package [Data] = $R_{enc}\big(Data, IV, Frag(Key, S_s)\big)$;

9: String Device = getDevice();

10: Package [DeviceID] = $R_{enc}\big(Device, IV, Frag(Key, S_s)\big)$;

11: Package $[S_{salt}] = RSA_s(S_s, Publickey)$;
12: String Token = getToken();
13: Package [Token] = Tokenize (Token, $S_s$);
14: String LSignature = getLSignature();
15: Package [Chain] = Tokenize (LSignature, $S_s$);
16: String Time = getNew();
17: String Signature = $C_{sig}(S_s + Date(Year) + Request + Data + Device + Token + Seq + Time, S_s)$
18: Package [Signature] = Signature;
19: Package [Time] = Time;
20: String JPackage = JSON Encode (Package);
21: JPackage = $S_{enc}(JPackage, IV, KEY)$;
22: String Result = Null;
23: Result = SendRequest (Endpoint, JPackage);
24: If (Result is Null OR Empty)
25: {
26: goto Retry;
27: }
28: Else
29: {
30: String Response = $S_{dec}(Result, IV, KEY)$;
31: Array ResPackage = JSONDecode (Response);
32: If (ResPackage is Null OR Empty)
33: {
34: goto Retry;
35: }
36: Else
37: {
38: String ResData = $R_{Dec}(ResPackage\ [Data], IV, Frag\ (KEY, S_s))$;
39: String ResExrta = $R_{Dec}(ResPackage\ [Extra], IV, Frag\ (KEY, S_s))$;
40: String ResToken = $R_{Dec}(ResPackage\ [Token], IV, Frag\ (KEY, S_s))$;

41: String ResTime = ResPackage [Time];
42: String ResSeq = ResPackage [Seq];
43: String ResDeviceID = ResPackage [DeviceIDVerify];
44: String ResSignature = ResPackage [Signature];
45: String VerSig = $S_{sig}(S_s + ResData + Date(Year) + ResSeq + ResToken + ResTime + Request + ResExtra + Device, S_s)$;
46: String VerDiv = $D_{sig}(Device + S_s, ResToken)$;
47: If (VerDiv Equals ResDevice AND VerSig Equals ResSignature)
48: {
49: If (Now+24H>ResTime>Now-24H)
50: {
51: If (Seq Equals ResSeq)
52: {
53: If (ResToken is Not Empty AND ResToken Not Equals Token)
54: {
55: If (SendExchange Equals True)
56: {
57: SetToken (ResToken);
58: }
59: }
60: Set LSignature (Signature);
61: //Other Actions Here
62: }
63: Else
64: {
65: Error (Seq id Invalid);
66: }
67: }
68: Else
69: {
70: Resync Time ();

```
71:                 If (Now+24H<ResTime<Now-24H)
72:                 }
73:                 Error (Invalid Response Package Time );
74:                 goto Retry;
75:             }
76:         }
77:     }
78:     Else
79:     {
80:         Error (Invalid Signature OR Device);
81:         goto Retry;
82:     }
83:     Flush (Package, Request, S_s, Device, Token, LSignature, time, Signature);
84:     Flush (JPackage, Result, Response, ResPackage, Data, ResToken);
85:     Flush (ResTime, ResSeq, ResDeviceID, ResSignature, VerSig, VerDiv);
86:     If (According to Server Response, There is need to Re-Authenticate)
87:     {
88:         Goto ReAuth;
89:     }
90:     Else
91:     {
92:         Do Some Actions Here
93:     }
```

### 7.3 شبه کدهای ارتباطات گره با گره – قسمت درخواست

```
1: Array Package = {};
2: String Request = “TestResource”;
3: Package [Request] = Request;
4: String data = “My Data”;
5: Int Seq =1;
6: Package [Seq] = Seq;
```

7: String $S_c$ = HighRandom();

8: Package [Data] = $OR_{enc}\big(Data, IV, Frag(Key, S_c)\big)$;

9: Package [$C_{salt}$] = $RSA_c\big(S_c, PeerKey_{Target}\big)$;

10: String Time = getNow();

11: Package [Time] = Time;

12: Array MyRoute = {Endpoint: EP, PKey: PK , PeerID: PID};

13: String Route = JSONEncode(MyRoute);

14: String PeerID = getMyIDFromTable();

15: Package [MyRoute] = Route;

16: Package [PeerID] = PeerID;

17: String Signature = $A_{Sig}(S_c + Date(Year) + Request + Data + PeerID + Time + Seq + Route + S_c)$;

18: String JPackage =JSONEncode (Package);

19: JPackage = $S_{enc}(JPackage, IV, KEY)$;

20: String Result = Null;

21: Result = SendPeerRequest ($Endpoint_{Target}$, $JPackage$);

22: If (Result is Null OR Empty)

23: {

24: goto Retry;

25: }

26: Else

27: {

28: String Response = $S_{dec}(Result, IV, KEY)$;

29: Array ResPackage = JSONDecode (Response);

30: If (ResPackage is Null or Empty)

31: {

32: Goto Retry;

33: }

34: Else

35: {

36: String $S_p = RSA_p(ResPackage\ [Sodium], PrivateKey_{Own})$;

String ResData = $TR_{dec}(ResPackage[Data], Frag(S_p, S_c))$;

String ResSeq = ResPackage [Seq];

String ResTime = ResPackage [Time];

String ResMyRoute = $TR_{dec}(ResPackage[MyRoute], Frag(S_p, S_c))$;

Array RTable = JSONDecode (MyRoute);

String PeerIDVerify = ResPackage [PeerIDVerify];

String RSignature = ResPackage [Signature];

String RealSig = $B_{Sig}(S_c + Date\ (Year) + Request + Data + S_p + PeerID + Time + MyRoute, S_c)$;

String RealPeer = $P_{Sig}(PeerID + S_c, S_p)$;

```
If (RealPeer NOT Equals PeerIDVerify)
  {
    Error (PeerID is Invalid);
    goto Retry;
  }
If (RealSig NOT Equal RSignature)
  {
    Error (Signature is Invalid);
    goto Retry;
  }
If (Now+24H>ResTime>Now-24)
  {
    ResyncTime();
    If (Now+24H<ResTime<Now-24H)
        {
          Error (Time is Invalid;
          goto retry;
        }
    }
If (Seq Not Equal ResSeq)
    {
```

```
67:             Error (Seq# is Not as Expected);
68:             goto Retry;
69:             }
70:         If (RTable is NOT Null OR Empty)
71:             {
72:             SetRoute (RTable);
73:             }
74:         If (ResData is Valid AND NOT Null OR Empty)
75:             {
76:             Other Actions Here;
77:             }
78:             Else
79:             {
80:             Error (Data is Invalid OR Empty);
81:             }
82:       }
```

### 7.4 شبه کدهای ارتباطات گره با گره – قسمت درخواست

```
1: Array RPack = {};
2: String Input = Null;
3: Input = getInput(.);
4: If (InputMethod is NOT Valid OR Input is Null or Empty)
5:   {
6:    Error (Input is Invalid or Empty);
7:   }
8: Try
9:   {
10:  Input = S_dec(Input, IV, Key);
11:  RPack = JSONDecode (Input);
12:  If (All Input Fields are NOT Exact AND NOT Exist even Empty Strings)
13:    {
```

14: Error (Input is Invalid OR Empty);
15: }
16: String Request = RPack[Request];
17: String Signature = RPack [Signature];
18: String RData = RPack [Data];
19: String $C_{Salt}$ = RPack [$C_{Salt}$];
20: String $S_c$ = $RSA_c$ ($C_{Salt}, PeerKey_{Own}$);
21: String Data = $OR_{dec}(RData, IV, Frag(KEY, S_c))$;
22: String PeerID = $OR_{dec}(RPack[PeerID], IV, Frag(KEY, S_c))$;
23: String Seq = RPack [Seq];
24: String MyRoute = $OR_{dec}(RPack[MyRoute], IV, Frag(KEY, S_c))$;
25: Array Route = JSONDecode (MyRoute);
26: String Time = RPack [Time];
27: }
28: Catch { Error (Unknown Error Occurred);}
29: String SigVerify = $A_{Sig} = (S_c + Date(Year) + Request + Data + PeerID + Time + Seq + MyRoute, S_c)$;
30: If (SigVerify NOT Equal Signature)
31: {
32: Error (Signature is Invalid);
33: }
34: If (Now+24H>Time>Now-24H)
35: {
36: Resync Time;
37: If (Now+24H<Time<Now-24H)
38: {
39: Error (Time is Invalid)
40: }
41: }
42: If (PeerID NOT Valid AND Requested Resource NOT Available)
43: {

44: Error (Invalid Request);

45: }

46: If (Route is NOT Null OR Empty)

47: {

48: SetRoute (Route);

49: }

50: If (Request is Valid AND Exist)

51: {

52: String Res = ProcessMethod (Request);

53: Other Actions Here;

54: Array RespPack = {};

55: String $S_p$= HighRandom();

56: String PeerIDVerify = $P_{Sig}(PeerID + S_c + S_p)$;

57: String Sodium = $RSA_p(S_p, PeerKey_{Target})$;

58: RespPack [Data] = $TR_{enc}(Res, IV, Frag(Key, S_p))$;

59: RespPack [Sodium] = Sodium;

60: RespPack [Seq] = Seq;

61: RespPack [PeerIDVerify] = PeerIDVerify;

62: String Time = getNow();

63: RespPack [Time];

64: Array MyRoute = {Endpoint: EP, PKey: PK, PeerID:PID};

65: String Route = JSONEncode (MyRoute);

66: RespPack [Signature] = $B_{Sig}(S_c + Date(Year) + Request + Res + S_p + PeerID + Seq + Time + Route, S_c)$;

67: RespPack [MyRoute] = $TR_{enc}(Route, IV, Frag(Key, S_p))$;

68: String RespData = JSONEncode(RespPack);

69: RespData = $S_{enc}(RespData, IV, Key)$;

70: SendResponse (EndPoint, RespData);

71: }

72: Else

73: {

```
74:     Error (Request is invalid);
75:   }
```

### 7.5 شبه کدهای ارتباطات سیستم سیگنال – سمت سرور ابری

```
1:   Array SigPack = {};
2:   String Input = Null;
3:   Input = getSignal(.);
4:   If (InputMethod is NOT Valid OR Input is Null OR Empty)
5:     {
6:       Error (Input is Invalid or Empty);
7:     }
8:   Try
9:     {
10:      Input = S_dec(Input, IV, kEY);
11:      Sigpack = JSONDecode (Input);
12:      If (All Input Fields are NOT Exact AND NOT Exist even Empty Strings)
13:        {
14:          Error (Input is Invalid OR Empty);
15:        }
16:      String Identifier = SigPack [Identifier];
17:      String Data = SigPack [Data];
18:      String DeviceID = FindDeviceIDbyIdentifier (Identifier);
19:      If (DeviceID is Null OR Empty)
20:        {
21:          Error (Cannot Find This Device or Session);
22:        }
23:      String Token = getTokenbyDeviceID (DeviceID);
24:      String LSignature = get LSignaturebyDeviceID (DeviceID);
25:      Data = N_dec(Data, IV, Frag(KEY, Token, LSignature));
26:      String Time = SigPack [Time];
27:      String Signature = SigPack [Signature];
```

28: String SigVer = $N_{sig}(DeviceID + LSignature, Token)$;
29: If (Signature NOT Equals SigVer)
30: {
31: Error (Signature is Wrong);
32: }
33: If (Now+24H<Time<Now-24H)
34: {
35: Error (Time is Invalid, Please Resync Time and Retry);
36: }
37: String flag = Process Signal (DeviceID, Data);
38: SignalResponse (Endpoint, flag);
39: }
40: Catch { Error (Unknown Error Occurred); }

### 7.6 شبه کدهای ارتباطات سیستم سیگنال – سمت سرور ابری

1: Array Signal = {};
2: String LSignature = getLSignature ();
3: String Token = getToken ();
4: String Device = getDevice();
5: String Identifier = $N_{Sig}(Device + LSignature, Token)$;
6: String Data = “Everything You Want”;
7: String Time = Now();
8: String Signature = $M_{Sig}$ $(Device + Data + Time, Token)$;
9: Signal [Identifier] = Identifier;
10: Signal [Data] = $N_{enc}(Data, IV, Frag(Token, LSignature))$;
11: Signal [Time] = Time;
12: Signal [Signature] = Signature;
13: String SignalData = JSONEncode (Signal);
14: SignalData = $S_{enc}$(SignalData,IV,KEY);
15: String Response = SignalRequest (Endpoint,SignalDate);
16: If (Response is Null or Empty)

```
17:   {
18:     Goto Retry;
19:   }
20:  If (Response Equal Zero)
21:   {
22:     Nothing to Do, So Retry after a period of Time;
23:   }
24:  If (Response > 0)
25:   {
26:     There is an Update, Let's get it from server, Type of request that should be made is related
        to this Positive Number;
27:     }
28:  If (Response < 0)
29:   {
30:     There is an Error that related to this Negative Number, The correct action should be done
        based on Error Handler;
31:   }
```

## 7.7 توابع استفاده شده در شبه کد ها

توابع استفاده شده در شبه کدهای فوق در جدول زیر توضیح داده شده است:

جدول 1-7 لیست توابع استفاده شده در شبه کد ها - قسمت اول

| | |
|---|---|
| Error (String Message) | تابعی برای نمایش یا بازگرداندن متن یا مولفه خطا در صورت بروز |
| JSONEncode (Array Object) | تابع تبدیل آرایه به رشته JSON برای تبادل |
| JSONDecode (String Input) | تابع تبدیل رشته JSON به آرایه یا شی برای پردازش |
| ResyncTime() | تابع تنظیم و بهنگام سازی مجدد مبدا زمانی حال به صورت داخلی یا سیستمی |
| SetRoute(Array RO) | ذخیره شی RO در جدول مسیرها و کلیدها برای تبادلات نظیر به نظیر |
| ProcessMethod | پردازش متد یا روال داخلی که توسط طراح سیستم نوشته شده و منابع درخواست شده که نتیجه را بر می گرداند |
| HighRandom | تابع گرفتن مقدار بسیار تصادفی |
| getNow,getTime | گرفتن زمان حال به صورت Time Stamp از سیستم |

جدول 2-7 لیست توابع استفاده شده در شبه کد ها - قسمت دوم

| | |
|---|---|
| Frag | ترکیب دو مقدار به صورتی که کقدار آرگومان دوم با مقدار آرگومان اول اجتماع گرفته شده و سپس N کاراکتر جدا می گردد. لذا ارزش یا اهمیت ارگومان با مقدار دوم است |
| SendRequest | ارسال درخواست به هر شکل و هر پروتکل که در سرور ابری تعبیه شده است |
| SendResponse | ارایه پاسخ درخواست ارسال شده به همان گره درخواست کننده |
| getMyIDFromTable | گرفتن مقدار MYID از جدول مسیرها (Peer Table) متناسب با گره مورد تبادل |
| getCandidateToken | گرفتن توکن کاندید شده برای تعویض از پایگاه داده که متناسب با گره مورد تبادل است |
| PutCandidateTokenAsToken | ذخیره توکن کاندید به عنوان توکن اصلی در پایگاه داده |
| SetIdentifier | ذخیره مقدار Identifier مرتبط با گره درخواست دهنده در سرور برای تبادلات سیستم Signal |
| SendPeerRequest | ارسال درخواست گره به گره دیگر به صورت پروتکل ارتباطی یکسان تعریف شده در هر دو |
| SendPeerResponse | پاسخ به درخواست گره مورد تبادل با پروتکل ارتباطی یکسان تعریف شده در هر دو |
| Tokenize | امضا از ترکیب دو مقدار با استفاده از تابع SHA1 می باشد |
| getLSignature | گرفتن آخرین امضا درخواست نسبت به گره مورد تبادل |
| getLToken | گرفتن آخرین توکن ذخیره شده در پایگاه داده |
| ForceReAuth | اقدام اجباری ابطال احراز هویت و جلسه و الزام به احراز هویت مجدد |
| SetLSignature | تنظیم مقدار به عنوان آخرین امضا درخواست نسبت به گره مورد تبادل |
| RecordSSalt | ذخیره $S_S$ یا همان نمک درخواست |
| SetToken | ذخیره مقدار جدید به عنوان توکن |
| getToken | گرفتن توکن اصلی ذخیره شده |
| Date | گرفتن عدد هر عنصر از تاریخ حال |
| getDevice | گرفتن شماره منحصر به فرد دستگاه/جلسه |

لازم به ذکر است کد های قابل استفاده و کاربردی در آدرس زیر[51] به صورت متن باز و رایگان بارگذاری گردیده است و پروتکل در اپلیکیشن اندرویدی افتا پارس اجرا شده و در حال استفاده می باشد.

# 8 کارهای مشابه

سال ها است پروتکل هایی جهت امن سازی تبادل اطلاعات در شبکه تولید و ارائه شده است. با این حال هرکدام برای استفاده خاصی تعبیه شده و یا یک مجموعه کاملی که بتوان آن را راهکار کاملی به حساب آورد ارائه نگردیده است. از طرفی دیگر راهکارهایی که به عنوان یک راهکار جامع و کامل ارائه شده، به دلیل پیاده سازی های سنگین و دشوار یا سازگار نبودن با بسیاری از حالات اتصال و انواع احراز هویت منسوخ شده یا عملا به صورت عمومی مورد استفاده و بهره برداری قرار نگرفته است. در ادامه به چند راهکار مشابه پروتکل پیشنهادی اشاره کرده و نقاط ضعف آن ها در برابر راهکار پیشنهادی را توضیح می دهیم.

## 8.1 پروتکل HTTP Secured

یک پروتکل امنیتی بر پایه پروتکل SSL [52] که امروزه به نام TLS [53] هم شناخته می شود. این پروتکل ترافیک بین دو گره یا سرور و گره را با الگوریتم ها و مکانیزم های تبادل کلید در محیط نا امن، امن سازی کرده و تایید صحت می کند. اما نکته قابل توجه این است که این پروتکل هیچ گاه یک راهکار کامل نبوده است. زیرا علاوه بر نقایص و مشکلات منطقی در پیاده سازی، همچنان امکان شنود با جعل سندها و گواهی های امنیتی یا به اصطلاح Spoofing وجود دارد. همچنین این پروتکل با کنترل کننده ها و پاسخ دهنده ها یکپارچه نشده و نمی توان به امنیت داده ها از لحاظ اینکه با استفاده از این الگوریتم شنود یا دستکاری نشوند مطمئن بود. در این پروتکل چنانچه گواهی ها جعل یا سرقت شوند، تایید صحت انجام نشده و علاوه بر اینکه شنود ممکن است صورت گرفته باشد، می تواند از رسیدن اطلاعات به گره دیگر جلوگیری به عمل آورد و در نتیجه با از دست دادن داده ها مواجه بشویم. بسیاری از حملات بر پایه شبکه نیز وجود دارند که این پروتکل امکان و قابلیت شناسایی و رفع آن ها را ندارد و به طور جداگانه بایستی امکانات دیگری در این رابطه پیاده سازی گردند.

## 8.2 پروتکل MTProto

یکی از پروتکل ها و قواعد مجتمع و پیچیده برای امن سازی تبادل داده ها و اطلاعات در شبکه است که توسط تلگرام برای اولین بار مطرح گردید و برای پیام رسان تلگرام مورد استفاده قرار گرفت. این پروتکل را می توان یک پروتکل کامل و جامع به حساب آورد، اما این پروتکل در نسخه سرور خود منبع باز نیست و کنترل کننده ها و پاسخ دهنده ها ندارد. از این رو صرفا جهت رمزنگاری داده و ارسال آن در کانال های ناامن و ناشناخته کاربرد دارد. از طرف دیگر، پیچیدگی

---

[51] https://github.com/nimix3/AngelsGate
[52] Secure Sockets Layer
[53] Transport Layer Security

پیاده سازی نسبتا بالایی دارد و از آنجا که از الگوریتم های غیررسمی در آن استفاده شده به آسانی قابل بکارگیری نیست. همچنین این پروتکل صرفا جهت استفاده در پیام رسان تلگرام تعبیه شده و الگوریتم های آن مختص استفاده در حالت مبتنی بر جلسه و با معماری خاص تلگرام می باشند و چندان قابلیت سازگاری با انواع و اشکال ارتباطات و داده ها را ندارد.

# 9 منابع

1. Job, J., et al., A modified secure version of the Telegram protocol (MTProto). 2015 Ieee International Conference on Electronics, Computing and Communication Technologies. 2015.

2. Jakobsen, J., C. Orlandi, and Acm, On the CCA (in) Security of MTProto. Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices. 2016. 113-116

3. Stallings, W., Cryptography and network security: principles and practice. 2017: Pearson Upper Saddle River, NJ.

4. Lindell, Y. and J. Katz, Introduction to modern cryptography. 2014: Chapman and Hall/CRC

5. Wu, L., et al., *Secure Key Agreement and Key Protection for Mobile Device User Authentication.* IEEE Transactions on Information Forensics and Security, 2019. **14**(2): p. 319-330

6. Mukherjee, B., et al., Flexible IoT security middleware for end-to-end cloud–fog communication. Future Generation Computer Systems, 2018. 87: p. 688-703

7. Stallings, W., Network Security Essentials: Applications and Standards. 2016: Pearson