Autoridade Certificadora Dinâmica para Redes Ad Hoc Móveis*

Fernando Carlos Pereira[1†], **Joni da Silva Fraga**[1†]
Adriana Elissa Notoya[1], **Ricardo Felipe Custódio**[2]

[1]Programa de Pós-Graduação em Engenharia Elétrica (PPGEEL)
Universidade Federal de Santa Catarina (UFSC)

[2]Laboratório de Segurança em Computação (LabSEC)
Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brasil

{fernando,fraga,elissa}@das.ufsc.br, custodio@inf.ufsc.br

Abstract. *In this paper, we propose a dynamic, self-adaptable and intrusion tolerant certificate authority, designed to operate in MANETs. This CA is composed by a set of mobile devices presents in a MANET, which form a group of servers that support the CA functions. The architecture is based on dynamic systems model and was conceived to manage changes in the membership of servers group, decurrent of mobility or the non-availability, even though transitory, of part these servers. In reason of these changes, this paper introduce an algorithmic base that allows the CA reconfiguration, guaranteeing the availability and the inviolability of the certification service.*

Resumo. *Este trabalho propõe uma autoridade certificadora dinâmica, auto-adaptável e tolerante a intrusões, projetada para operar em MANETs. Esta AC é disposta sobre um conjunto de dispositivos computacionais móveis presentes em uma MANET, formando um grupo de servidores que suportam suas funcionalidades. O modelo de sistemas dinâmicos determina que a mesma esteja sujeita a variações na sua composição. A arquitetura foi concebida para gerenciar estas mudanças na composição do grupo de servidores, decorrentes da mobilidade ou da indisponibilidade, mesmo que momentânea, de parte destes servidores. Em razão destas mudanças, este trabalho introduz uma base algorítmica que permite a reconfiguração da AC, garantindo a disponibilidade e a inviolabilidade do serviço de certificação.*

1. Introdução

Ambientes distribuídos P2P caracterizados em redes *ad hoc* móveis (MANETs - *mobile ad hoc network*) devem executar suas aplicações usando recursos distribuídos de maneira descentralizada. O grande desafio nestes ambientes é manter o progresso da aplicação mesmo diante do que é identificado na literatura como *churn* [Godfrey et al. 2006]: dispositivos móveis entram e saem do sistema em tempos arbitrários.

Modelos de *sistemas dinâmicos* [Aguilera 2004, Mostefaqui et al. 2005] estão sendo introduzidos na literatura no sentido de garantir a evolução de aplicações distribuídas nestes ambientes. Para garantir o progresso de uma computação, alguns autores sustentam a necessidade de que um certo número mínimo de nós (dispositivos

*Trabalho apoiado pelo CNPq (processo 550114/2005-0).
†Bolsista CNPq.

móveis) permaneça tempo suficiente em sistemas compostos por dispositivos móveis. Em [Mostefaqui et al. 2005] é introduzida a noção de estabilidade terminal (*eventual stability*) que é centrada sobre um parâmetro α, definindo o número de dispositivos (nós) necessários para compor simultaneamente o sistema por um período suficiente de modo a garantir a evolução da computação distribuída. Nenhum nó permanece para sempre no sistema, mas um número limite α de nós tem que estar presente sempre para que o sistema avance.Este parâmetro captura então a estabilidade terminal de um sistema dinâmico.

Este artigo apresenta nossos primeiros esforços com sistemas dinâmicos. Assumimos, neste texto, MANETs como sistemas constituídos por dispositivos móveis que se organizam dinamicamente. Os grandes desafios envolvidos no projeto de aplicações distribuídas em MANETs giram em torno do *churn* e da dificuldade de centralizar funções em alguns nós destas redes. Outros aspectos limitadores nestes ambientes são as restrições de energia, de comunicação e ainda a reduzida capacidade computacional em alguns tipos de dispositivos móveis. Entre as premissas que orientaram nossos trabalhos estava a necessidade de desenvolver algoritmos sem o uso de uma entidade ou estrutura central.

Estes novos ambientes dinâmicos abrem caminho para novas classes de aplicações. Centramos nossos esforços no desenvolvimento de uma Autoridade Certificadora Dinâmica (ACD), a qual tem as funções de certificação de chaves criptográficas e de gerenciamento da própria arquitetura desempenhadas por um grupo de servidores. Estas funções e serviços envolvem a execução de algoritmos distribuídos nestes ambientes dinâmicos e considerando possíveis intrusões em servidores da ACD. A criptografia de limiar é usada para dificultar a disponibilidade da chave privada da ACD. A inovação que introduzimos neste artigo é que a composição dos servidores da ACD é não fixa, variando conforme as imposições características destes ambientes. As variações de sua composição (n) e do número de intrusões que pode sofrer (t) dependem do parâmetro α que garante a estabilidade terminal da ACD.

O artigo está organizado da seguinte maneira. Na seção 2 é apresentado o modelo da ACD. A seção 3 apresenta a base algorítmica proposta. A seção 4 apresenta os protocolos de criptografia de limiar usados. A seção 5 faz considerações sobre a ACD e os algoritmos propostos. A seção 6 apresenta os trabalhos relacionados. E, por fim, a seção 7 apresenta as conclusões do trabalho.

2. A Autoridade Certificadora Dinâmica – ACD

A ACD é constituída por um grupo $\mathcal{S}$ de n nós servidores, os quais desempenham através de cooperação as funções que envolvem a execução de protocolos que controlam ou apenas usam as *chaves do serviço* ACD, as quais correspondem a um par de chaves criptográficas pública e privada. A chave pública (ku_{acd}) é conhecida por todos os nós da rede e usada na verificação das assinaturas da ACD em certificados emitidos pela mesma. Já a chave privada correspondente (kr_{acd}), usada para gerar as assinaturas da ACD, possui garantida a sua inviolabilidade através da criptografia de limiar. Ou seja, a kr_{acd} não está disponível em momento algum no sistema. O seu uso se dá através de um conjunto $\mathcal{K}$ de *chaves parciais*[1] ($|\mathcal{K}| = n$, $\mathcal{K} = \{k_1, ..., k_n\}$) derivadas da kr_{acd} por meio de técnicas de *compartilhamento de segredo* [Shamir 1979]. Qualquer operação com a kr_{acd} só é possível através do uso de ao menos $t + 1$ destas chaves parciais ($t \leq \frac{n-1}{2}$). No modelo

[1]Usualmente, em protocolos de criptografia de limiar a designação adotada é *"shares"* (partes). A denominação *chaves parciais* foi adotada por tornar o texto melhor compreensível.

proposto, cada uma destas chaves parciais é entregue a um servidor específico em $\mathcal{S}$, o que implica que a execução de qualquer protocolo que envolva a chave kr_{acd}, seja efetivada por um subgrupo $\mathcal{V}$ de ao menos $t+1$ servidores corretos ($\mathcal{V} \subset \mathcal{S}$). A ACD tolera até t servidores faltosos ou maliciosos. Assume-se, portanto, que o parâmetro α, representante da estabilidade terminal da ACD, corresponde a $2t+1$ servidores e que $|\mathcal{V}| \geq 2t+1$.

A composição de $\mathcal{S}$ está sempre sujeita a mudanças devido às possíveis entradas e saídas de servidores, que são eventos normais em sistemas dinâmicos. Portanto, em um dado instante, conhecer a quantidade e a identificação dos servidores presentes em $\mathcal{S}$ nem sempre é possível. Estas informações são conhecidas por completo apenas na fase de iniciação da ACD (seção 3.1), quando os servidores são configurados previamente com a informação sobre a composição inicial de $\mathcal{S}$. Uma vez iniciada a ACD, a composição de $\mathcal{S}$ passa a estar sujeita a alterações e os algoritmos usados são executados fazendo uso de estimativas desta composição obtidas em períodos de *sincronização*. Nestes períodos são executadas funções de gerenciamento que estimam a composição do grupo de servidores naquele momento e renovam as chaves parciais do sistema. O término de um período de sincronização sinaliza o início de um período que chamamos de *época*.

Uma *época* é um período de tempo onde um determinado conjunto $\mathcal{K}$ de chaves parciais e uma estimativa da composição da ACD permanecem em vigor (ou seja, seus valores permanecem inalterados). Cada *época* é identificada por um contador ε que inicia em 0 e é incrementado após cada novo período de sincronização. As chaves parciais pertencentes a uma *época* são incompatíveis com chaves parciais de outras *épocas*. Portanto, para um adversário obter êxito no comprometimento da chave kr_{acd}, o mesmo deve obter $t+1$ chaves parciais pertencentes a uma mesma *época*. O fim de uma *época* é controlado pelos servidores em seus relógios locais, sem a necessidade de sincronização de relógios.

O controle de servidores faltosos é feito por meio de *listas negras*, as quais relacionam os servidores faltosos. Cada servidor gerencia a sua própria lista negra. A inclusão de um servidor nesta lista é feita mediante o recebimento mensagens acusações enviadas por servidores ou por clientes. Nestas mensagens de acusação devem estar contidas as informações incorretas enviadas pelo servidor acusado. O servidor presente em mais do que t listas negras é excluído de $\mathcal{S}$ no próximo período de sincronização, e só poderá voltar a integrar a ACD em caso específico a ser apresentado em seção posterior.

2.1. Premissas do Suporte de Comunicação

O modelo de comunicação adotado assume que, havendo densidade de nós suficiente, o protocolo de roteamento utilizado na rede *ad hoc* assegura para cada nó a disponibilidade de rotas para ao menos $t+1$ servidores de $\mathcal{S}$, e que a informação sobre a distância (número de saltos) de cada rota é disponível. Estas premissas são adotadas para garantir a conectividade entre os servidores em $\mathcal{S}$ e o acesso aos serviços da ACD pelos demais nós. No que refere-se às mensagens enviadas durante a execução dos algoritmos e protocolos do sistema, *loops* de envio são evitados inserindo identificadores únicos nas mensagens. Já a autenticidade destas é obtida através das assinaturas digitais que assume-se estarem presentes em todas as mensagens. Com base nestas premissas e a fim de tornar mais simples os algoritmos propostos, são omitidos os passos de verificação de unicidade, integridade e autenticidade das mensagens envolvidas. Entretanto, estas ações são essenciais e, portanto, implícitas ao recebimento de quaisquer mensagens nos algoritmos e protocolos.

2.2. Formato do Certificado ACD

O formato de um *certificado ACD*[2] é semelhante ao do padrão X.509, porém não são compatíveis entre si. A incompatibilidade se dá devido ao valores dos campos $issued\_on$ e $expire\_on$, que na ACD são definidos em termos da época em que o certificado foi emitido e a época a partir da qual terá sua validade expirada. O uso de selos de tempo determinados pelo padrão X.509 implicaria na necessidade de sincronização de relógios entre os servidores, e conseqüentemente aumentaria a complexidade do serviço ACD. A chave pública presente no campo ku_i do certificado ACD é extraída da *credencial de autorização*[3] apresentada pelo titular do certificado. Já o campo $server$ recebe o valor 1 quando o titular é um nó servidor e 0 quando é um nó comum.

3. Base Algorítmica da ACD

A base algorítmica da ACD contempla a configuração inicial do sistema, através do estabelecimento da *época* 0, a entrada de novos dispositivos móveis na MANET, a emissão de certificados para dispositivos presentes na rede, e o gerenciamento da composição do grupo $\mathcal{S}$ de servidores e dos parâmetros de configuração do conjunto $\mathcal{K}$ de chaves parciais. Os algoritmos propostos nesta seção fazem uso dos protocolos criptográficos de *geração distribuída de chaves criptográficas* (`Thresh-Key-Gen`), de *assinatura baseada em limiar* (`Thresh-Sig`) e de *redistribuição de chaves parciais* (`Thresh-Redist`), os quais são descritos na seção 4.

3.1. Sincronização da *época* 0

Esta fase é conduzida de forma distribuída pelos servidores que formam o grupo $\mathcal{S}$ inicial, através do algoritmo 3.1. Este algoritmo visa estabelecer a chave pública ku_{acd} e o conjunto $\mathcal{K}$ de chaves parciais, configurando cada servidor i ($s_i \in \mathcal{S}$) com a respectiva chave parcial k_i, válida na *época* 0. O certificado auto-assinado da ACD ($cert_{acd}$) é também emitido e publicado nesta fase. A informação sobre a composição inicial de $\mathcal{S}$ é repassada aos servidores pelo administrador do sistema, o qual também atua na execução do protocolo `Thresh-Key-Gen` determinando o conjunto de servidores qualificados a fornecerem os valores utilizados na geração do conjunto de chaves parciais do serviço. O conhecimento prévio da composição de $\mathcal{S}$ nesta fase é necessário para evitar a formação de grupos disjuntos de servidores e, conseqüentemente, de mais do que um conjunto de chaves parciais do serviço.

Os dois primeiros passos do algoritmo 3.1 são usados pelo servidor s_i para anunciar a sua presença na rede através da difusão de uma mensagem $init$. Esta mensagem contém a credencial de autorização de s_i que o habilita a compor $\mathcal{S}$. O início efetivo da configuração dos servidores para a época 0 se dá quando houver um subgrupo mínimo de $2t + 1$ servidores presentes na rede (passo 7). Quando este número mínimo é alcançado, estes servidores cooperam na execução do protocolo `Thresh-Key-Gen` para que cada servidor s_i obtenha a sua chave parcial exclusiva k_i e a chave pública ku_{acd} correspondente. Cada servidor possuirá então esta visão $\mathcal{V}_i$ ($|\mathcal{V}_i| \geq 2t + 1$) do serviço ACD.

[2]Certificado ACD: $\langle id_{subject}, issued\_on, expire\_on, ku_i, date\_cred, server, id_{acd}, sig_{acd} \rangle$

[3]A *credencial de autorização* deve ser emitida por uma entidade na qual os servidores em $\mathcal{S}$ confiam, identificando de forma única o seu titular e, quando for o caso, indicando que o titular é um servidor. Esta credencial é exigida inclusive aos nós comuns e deve ser apresentada em outros dois momentos: na entrada na MANET (seção 3.2) e na solicitação de um certificado (seção 3.3)

Algoritmo 3.1 `ACD-Initialization()`

Server s_i **join** the network as part of its initialization process

1: $init_i \leftarrow \langle$*init*, id_{s_i}, $credential_{s_i}\rangle$
2: `broadcast`($init_i$)
3: $\tau \leftarrow$ **time()**
4: **while** (true) **do**
5: **upon** `receive` ($init_j$) **do**
6: $\mathcal{I}_i \leftarrow \mathcal{I}_i \cup \{init_j\}$
7: **if** $|\mathcal{I}_i| \geq (2t+1)$ **then**
8: `Thresh-Key-Gen`($administrator$, $\mathcal{S}$)
9: *exit-loop*
10: **end if**
11: **if** ((**time()** $-\ \tau$) $\geq init\_time\_out$) **then**
12: `broadcast`($init_i$)
13: $\tau \leftarrow$ **time()**
14: **end if**
15: **end while**
16: **if** (s_i *has the smallest id from set* $\mathcal{S}$) **then**
17: s_i coordinate the creation of $cert_{acd}$ and $epoch_\varepsilon$
18: **end if**
19: `send`($cert_{acd}$) to $administrator$
20: $cert_{s_i} \leftarrow$ `Issue-Certificate()`

Após obtida a chave ku_{acd} é preciso emitir o certificado da ACD ($cert_{acd}$), através do qual esta chave será divulgada (passo 17). Também é necessário emitir a lista de parâmetros $epoch_\varepsilon = \langle \varepsilon, t, n \rangle$, que será usada a posteriori pelos novos nós que entrarem na rede com a ACD já operacional. A emissão do certificado da ACD e desta lista é coordenada pelo servidor com o menor identificador i e envolve a execução do protocolo `Thresh-Sig`, gerando as assinaturas correspondentes da ACD. Estas emissões determinam o início da *época* 0 e sinalizam que o certificado $cert_{acd}$ e a lista $epoch_\varepsilon$ assinados pela ACD estão difundidos entre todos os servidores que participaram de todas as etapas da emissão destes documentos (tal afirmação decorre dos passos que compõe o protocolo `Thresh-Sig`). A publicação do certificado da ACD se dá por intermédio do administrador, o que explica o passo 19. No último passo do algoritmo, s_i utiliza o algoritmo de emissão de certificados (seção 3.3) para obter o seu próprio certificado ACD.

3.2. Entrada de um Novo Nó na MANET

Ao entrar na rede, um nó ($node_i$), sendo um novo servidor ou apenas um nó comum, deve localizar um subgrupo com ao menos $t+1$ servidores ACD para compor a sua *visão do grupo de servidores ACD* ($\mathcal{V}_{node_i}$). O mínimo de $t+1$ servidores é estabelecido para que $node_i$ que consiga posteriormente solicitar serviços de certificação à ACD, tendo a garantia de que a sua solicitação alcançará ao menos 1 servidor correto.

A localização de servidores se dá através de um mecanismo de busca em largura, onde o alcance da mensagem de busca enviada pelo novo nó é restrito aos nós presentes em um raio de alcance determinado por um certo número de saltos. O protocolo adotado na ACD consiste em $node_i$ enviar uma mensagem `discovery-req` aos seus vizinhos (número de saltos igual a 1) e aguardar pelo recebimento de respostas durante um determinado período de tempo. Cada servidor s_i que receber esta mensagem deve enviar ao $node_i$ uma mensagem $\texttt{discovery-rep}_{s_i} = \langle discovery\_rep, cert_{s_i}, epoch_\varepsilon \rangle$. No caso de $node_i$ ser um novo servidor, s_i o adiciona à sua própria visão $\mathcal{V}_{s_i}$, porém com um indicativo de que este novo servidor está inativo. Mediante o recebimento de mensagens de resposta, $node_i$ cria a sua visão $\mathcal{V}_{node_i}$ contendo uma relação dos identificadores

e certificados dos servidores que responderam à mensagem `discovery-req`. O $node_i$ também extrai das mensagens `discovery-rep` o parâmetro t, contido em $epoch_\varepsilon$, para determinar a cardinalidade mínima necessária para $\mathcal{V}_{node_i}$ ($|\mathcal{V}_{node_i}| \geq t+1$). É possível, entretanto, que servidores faltosos enviem mensagens $epoch_\varepsilon$ legítimas (assinadas pela ACD), porém desatualizadas por pertencerem a épocas passadas. Para evitar este problema, $node_i$ deve escolher, dentre as mensagens `discovery-rep` válidas recebidas, a que possui o conjunto de parâmetros $epoch_\varepsilon$ com o maior identificador de *época*.

Caso $node_i$ não receba qualquer resposta válida durante o período de espera ou a sua visão não alcançou o mínimo de $t+1$ servidores, o mesmo reenvia a mensagem `discovery-req`, mas com o alcance do *broadcast* aumentado em um salto em relação ao envio anterior.

Os nós servidores que entram durante uma época são rotulados como *inativos*, não participando da execução de serviços de certificação da ACD. Estes servidores somente deixam esta condição quando participarem de um processo de sincronização, onde serão integrados ao grupo $\mathcal{S}$ e receberão chaves parciais exclusivas, válidas na nova *época* que terá início.

3.3. Emissão de Certificado pela ACD

Todos os passos envolvidos na emissão de um certificado ACD são implementados no algoritmo 3.2. Inicialmente, um nó $node_i$ ao solicitar um certificado à ACD, escolhe na sua visão $\mathcal{V}_{node_i}$ um servidor para enviar a sua requisição, $\texttt{cert-req}_{node_i}$. Esta escolha é feita forma aleatória ou com base em critério de proximidade em relação a $node_i$. O servidor selecionado, ao receber a requisição, assume o papel de coordenador e conduz a emissão do certificado em cooperação com os servidores presentes na sua visão (e não na do seu cliente $node_i$). O servidor coordenador (s_c), por estar ativo, possui em sua visão um grupo de no mínimo $2t+1$ servidores ($\mathcal{V}_{s_c} \subseteq \mathcal{S}$, $|\mathcal{V}_{s_c}| \geq (2t+1)$).

Ao receber $\texttt{cert-req}_{node_i}$, s_c constrói o certificado $cert_{node_i}$ sem assinatura e o envia, junto com $\texttt{cert-req}_{node_i}$, em uma mensagem $\texttt{cert\_issue}_{node_i}$ para todos os servidores presentes em $\mathcal{V}_{s_c}$. Ao receber a $\texttt{cert\_issue}_{node_i}$, estes servidores cooperam junto com s_c na execução do protocolo `Thres-Sig`, o que resultará em $sig_{acd}(cert_{node_i})$, a assinatura da ACD sobre $cert_{node_i}$. No protocolo `Thres-Sig` cada componente s_i da visão $\mathcal{V}_{s_c}$ contribui com uma assinatura parcial feita usando sua chave parcial k_i. A assinatura final é obtida por s_c em um processo onde as assinaturas parciais são combinadas (seção 4.2). Ao obter $cert_{node_i}$ assinado, s_c envia o mesmo em uma mensagem $\texttt{cert-rep}_{s_c}$ ao cliente. Este certificado também é difundido entre todos os servidores que participaram da sua emissão.

É possível que o servidor s_c escolhido pelo cliente incorra em falhas maliciosas. Uma delas é o coordenador processar $cert_{node_i}$ corretamente com os seus pares, mas enviar dados espúrios em $\texttt{cert-rep}_{s_c}$ para o nó cliente. Neste caso, considerando que $\texttt{cert-rep}_{s_c}$ estará assinada por s_c, $node_i$ pode enviar uma acusação para todos os servidores de $\mathcal{V}_{node_i}$ divulgando $\texttt{cert-rep}_{s_c}$. Os servidores em $\mathcal{V}_{node_i}$ analisarão a acusação e se comprovarem a falta, incluirão s_c em suas listas negras. A outra falha maliciosa considerada é o servidor s_c se negar a processar a requisição recebida. Por este motivo, o cliente deve associar um intervalo de tempo (*time-out*) à $\texttt{cert-req}_{node_i}$, para controlar o tempo disponível para recebimento da resposta válida. Caso a resposta não seja recebida dentro do espaço de tempo delimitado pelo intervalo, o cliente envia a requisição para um

novo coordenador de $\mathcal{V}_{node_i}$, associando-a também a um novo intervalo. Este processo é repetido até que o cliente receba uma resposta válida. No pior caso o cliente terá que enviar a requisição para no máximo t servidores diferentes para atingir um servidor correto.

Algoritmo 3.2 `Issue-Certificate()`

Node $node_i$ (Client):
1: `cert-req`$_{node_i} \leftarrow \langle$*'request'*$, id_{n_i}, ku_{node_i}, credential_{node_i}\rangle$
2: $s_c \leftarrow$`chose`$(\mathcal{V}_{node_i})$ {*return the service coordinator*}
3: `send(cert-req`$_{node_i}$`)` to s_c
4: $\tau \leftarrow$ **time()**
5: **while** (true) **do**
6: **upon** receive `cert-rep`$_{s_c}$ **do**
7: **if** $cert_{node_i}$ is correct **then** *exit loop*
8: `accusation` $\leftarrow \langle accusation, id_{n_i}, id_{s_c},$`cert-rep`$_{s_c}\rangle$
9: `send(accusation)` to every server in $\mathcal{V}_{node_i}$
10: **if** $((\textbf{time()} - \tau) \geq reply\_time\_out)$ **then**
11: $s_c \leftarrow$`chose`$(\mathcal{V}_{node_i})$
12: `send(cert-req`$_{node_i}$`)` to s_c
13: $\tau \leftarrow$ **time()**
14: **end if**
15: **end while**

Coordinator s_c: **upon** receive `cert-req`$_{node_i}$ **do**:
1: **if** a valid $cert_{node_i}$ already exists **then**
2: `cert-rep`$_{s_c} \leftarrow \langle$*'reply'*$, cert_{node_i}\rangle$
3: `send(cert-rep`$_{s_c}$`)` to $node_i$
4: **else**
5: $cert_{node_i} \leftarrow$ informations derived from `cert-req`$_{node_i}$
6: `cert_issue`$_{node_i} \leftarrow \langle$*'cert_issue'*,`cert-req`$_{node_i}, cert_{node_i}\rangle$
7: `send(cert_issue`$_{node_i}$`)` to every server in $s_j \in \mathcal{V}_{s_c}$
8: $sig_{acd}(cert_{node_i}) \leftarrow$ `Tresh-Sig`$(cert_{node_i}, \mathcal{V}_{s_c})$
9: $cert_{node_i} \leftarrow \langle cert_{node_i} \mid sig_{acd}(cert_{node_i})\rangle$
10: `cert-rep`$_{s_c} \leftarrow \langle$*rep*$, cert_{node_i}\rangle$
11: `send(cert-rep`$_{s_c}$`)` to $node_i$
12: **end if**

Every $s_j \in \mathcal{V}$: **upon** receive `cert_issue`$_{node_i}$ **do**:
1: **if** a valid $cert_{node_i}$ already exists **then**
2: `cert-rep`$_{s_j} \leftarrow \langle$*rep*$, cert_{node_i}\rangle$
3: `send(cert-rep`$_{s_j}$`)` to s_c
4: **else**
5: `Tresh-Sig`$(cert_{node_i}, \mathcal{V}_{s_c})$
6: **end if**

3.4. Sincronização da ACD

A sincronização da ACD envolve a execução de procedimentos que visam tratar os aspectos dinâmicos de $\mathcal{S}$, os quais impactam sobre a segurança da chave kr_{acd}. Estes procedimentos são implementados pelo algoritmo 3.3 e consistem em integrar e configurar os novos servidores, detectar e excluir do grupo os servidores maliciosos, e criar um novo conjunto de chaves parciais $\mathcal{K}$ com base na nova composição de $\mathcal{S}$, originada pela entrada e saída de servidores. Em particular, os servidores inativos da época que está terminando que participarem da sincronização deverão estar ativos na próxima época, recebendo, portanto, uma chave parcial válida. A visão destes novos servidores que estavam inativos deve passar de maior que $t + 1$ para no mínimo $2t + 1$ servidores. Qualquer servidor (ativo ou inativo) que não conseguir participar de todas as etapas do algoritmo, entrará na nova *época* na condição de inativo e só poderá se recuperar e voltar a participar ativamente da ACD em um novo período de sincronização.

Análogo à iniciação da ACD, o processo de sincronização conta com a participação de uma *terceira parte confiável (*$\mathcal{TPC}$) através do algoritmo 3.4. A $\mathcal{TPC}$, que pode ser o próprio administrador, possui as funções exclusivas de determinar a nova composição do grupo de servidores, criando um grupo $\mathcal{S}'$, e determinar o conjunto de servidores qualificados que serão fontes dos valores envolvidos na geração das novas chaves parciais do sistema.

3.4.1. Início da sincronização

A fase de sincronização é iniciada após o decurso de um período de tempo d, que determina a duração de cada época. Os servidores somente aceitam participar de uma fase de sincronização em duas situações: (1) se constatado o decurso do prazo d; ou (2) se recebidos ao menos $t + 1$ mensagens `sync`, emitidas por outros servidores. O prazo d corresponde à duração de uma *época* que cada servidor controla em seu relógio local. Mensagens `sync` sinalizam o início do processo de sincronização de cada servidor. Este mecanismo é implementado nas linhas 1 a 5 do algoritmo de sincronização.

3.4.2. Criação da visão $view_i$ e determinação da composição $\mathcal{S}'$

Cada servidor s_i cria, através do procedimento `create_`$view_i$, a sua visão $view_i$ contendo os servidores dos quais recebeu mensagens $sync$ e a sua lista negra. A composição de $\mathcal{S}'$ é determinada pela $\mathcal{TPC}$ com base nas visões enviadas pelos servidores, conforme os passos contidos no algoritmo 3.4. O grupo $\mathcal{S}'$ substituirá o atual grupo $\mathcal{S}$ na nova época ($\varepsilon + 1$) e é com base na sua composição que será construído e disseminado o novo conjunto de chaves parciais $\mathcal{K}'$ no procedimento `redist_partial_keys`.

3.4.3. Redistribuição das Chaves Parciais

A redistribuição das chaves parciais é feita através do protocolo `Thresh-Redist`. Este protocolo é utilizado para adequar as chaves parciais de $\mathcal{K}$, geradas usando a atual configuração (*t, n*), à uma nova configuração (*t', n'*), originada pela entrada ou saída de servidores de $\mathcal{S}$. A execução do algoritmo deve ser feita por um subgrupo ω de $2t + 1$ servidores ($|\omega| \geq 2t + 1$). Este tamanho é assumido para haver garantia de terminação do protocolo, já que assume-se haver até t servidores maliciosos. Em um cenário onde não há servidores faltosos ou maliciosos e há garantia na entrega das mensagens, é possível compor ω com apenas $t + 1$ servidores. Entretanto, considerando o caso onde $t + 1 \leq \omega \leq 2t$, e a existência de faltas ou malícia, não seria possível garantir a terminação do protocolo.

A redistribuição das chaves parciais deve ser conduzida por um grupo de servidores que possuam informações da *época* atual e da próxima. A composição de ω será, portanto, determinada de forma distribuída a partir da intersecção dos grupos $\mathcal{S}$ e $\mathcal{S}'$ ($s_i \in \omega \mid s_i \in (\mathcal{S} \cap \mathcal{S}')$). Os outros servidores fora desta intersecção, mas pertencentes a $\mathcal{S}'$, atuarão como simples receptores de chaves parciais. Os servidores de ω devem ser provenientes de $\mathcal{S} \cap \mathcal{S}'$ pois devem ser capazes de realizar a redistribuição da chave kr_{acd} a partir de chaves parciais pertencentes ao conjunto $\mathcal{K}$, ao mesmo tempo que devem ser capazes de disseminar as novas chaves parciais entre os integrantes do grupo $\mathcal{S}'$. Caso $|\omega| < 2t + 1$, não é possível garantir o sucesso do algoritmo `Thresh-Redist`, o que

Algoritmo 3.3 `Synchronization()`

Require: Server $s_{i\{i\in\mathcal{S}\}}$ `receive` $(sync_{j\{j\neq i, j\in\mathcal{S}\}})$ or period d is elapsed
1: **upon** `receive` $(sync_j)$ **do**
2: $\mathcal{V}'_i \leftarrow \mathcal{V}'_i \cup \{sync_j\}$
3: **if** (period of d is elapsed) or $(|\mathcal{V}'_i| > t)$ **then**
4: `create_`$view_i(\mathcal{V}'_i)$
5: **end if**

procedure `create_`$view_i(\mathcal{V}'_i)$
1: $sync_i \leftarrow \langle sync, \varepsilon, id_{s_i}, cert_{s_i}\rangle$
2: `send`$(sync_i)$ to every $s_j \in \mathcal{V}_{s_i}$
3: **while** $(|\mathcal{V}'_i| \leq 2t+1)$ **do**
4: **upon** `receive` $(sync_j)$ **do**
5: $\mathcal{V}'_i \leftarrow \mathcal{V}'_i \cup \{sync_j\}$
6: **end while**
7: $view_i \leftarrow$`create_new_view` $(\mathcal{V}'_i, black\_list_{s_i})$
8: `send`$(view_i)$ to $\mathcal{TPC}$
9: **upon** `receive`$(\mathcal{S}')$ **do** {*due to algorithm* 3.4}
10: $\mathcal{V}_{s_i} \leftarrow \mathcal{S}'$
11: `redist_partial_keys` $(\mathcal{S}')$

procedure `redist_partial_keys` $(\mathcal{S}')$
12: $\omega \leftarrow \mathcal{S} \cap \mathcal{S}'$
13: **if** $s_i \in \omega$ **then**
14: `Thresh-Redist` $(t', \omega, \mathcal{TPC})$
15: **if** (s_i *has the smallest id from set* $\mathcal{S}$) **then**
16: s_i coordinate the creation of $epoch_{\varepsilon+1}$
17: `broadcast`$(epoch_{\varepsilon+1})$
18: **end if**
19: **end if**

pode resultar na impossibilidade de terminação do processo de sincronização. Neste caso pode ser necessário repetir o processo ou retomar o sistema por meio da interferência do administrador.

Algoritmo 3.4 `define_`$\mathcal{S}'()$

$\mathcal{TPC}$ **upon** `receive` $(view_j)$ **do:**
1: $views \leftarrow views \cup \{view_j[\mathcal{V}_j]\}$
2: $black\_lists \leftarrow black\_lists \cup \{view_j[black\_list_j]\}$
3: **if** $(|view| \geq 2t+1)$ **then**
4: **for all** id_{s_j} that appear in **at least** $(t+1)$ $views$ and appear in **at most** (t) $black\_lists$ **do**
5: $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{id_{s_j}\}$
6: **end for**
7: **end if**
8: `broadcast`(S')

4. Criptografia nos Algoritmos Propostos

Nesta seção são apresentados os protocolos criptográficos empregados nos algoritmos da ACD. Estes protocolos utilizam criptografia de limiar, de modo que as operações criptográficas são executadas por um grupo de servidores que cooperam entre si através da execução de operações parciais, a fim de alcançar um resultado final válido na operação final. O criptossistema usado é baseado em *curvas elípticas* [Koblitz et al. 2000], o que permite usar chaves criptográficas menores, consumir menos recursos computacionais nas operações criptográficas e oferecer uma segurança similar à oferecida por outros sistemas de chave pública tradicionais[4].

[4]Outros sistemas criptográficos são os formados pelo RSA ou pelo DSA, por exemplo

Foram necessárias alterações para que estes protocolos se ajustassem à ACD. Estas alterações envolvem especificamente: (i) a delegação para um coordenador da tarefa de determinar o conjunto de servidores qualificados que serão fontes dos valores envolvidos na geração de chaves e assinaturas nos protocolo; e (ii) a inclusão dos servidores faltosos nas listas negras dos demais servidores.

A terminologia original dos protocolos foi substituída pela adotada na ACD. Entretanto, há alguns termos usados que são próprios da criptografia baseada em curvas elípticas e que devem ser apresentados. O termos T e T' denotam pontos definidos sobre os campos $GF(q)$ e $\mathcal{G}$, respectivamente, onde q é um número primo e $\mathcal{G}$ possui ordem p. O operador $\oplus$ representa a soma de pontos com módulo fixo ($mod\ p$), e $\sum^{\oplus}$ denota o somatório de pontos usando o operador $\oplus$. Conceitos mais específicos e técnicas para geração destes parâmetros são encontrados em [Tang et al. 2005, Koblitz et al. 2000].

4.1. Geração Distribuída de Chaves

Em protocolos de *geração distribuída de chaves* [Pedersen 1991, Tang et al. 2005] a chave pública do par é calculada por cada integrante do grupo, já a chave privada não é construída em momento algum e, ao término do protocolo, cada servidor possui apenas uma das partes desta. A construção da chave privada permanece possível, mas para que a sua segurança e confidencialidade sejam mantidas, as operações que necessitam do seu uso são executadas através de operações parciais feitas usando apenas as partes da chave (chaves parciais). A vantagem desta abordagem é que nenhuma entidade conhece a chave privada ou tem controle total sobre ela, sendo este controle exercido de forma compartilhada pelos integrantes do grupo. A desvantagem é que estes protocolos são mais complexos e exigem uma quantidade maior de trocas de mensagens e de processamento pelos os nós em relação à geração centralizada das chaves. Na ACD, a geração distribuída das chaves ku_{acd} e kr_{acd} é feita pelo protocolo 4.1, que é baseado em [Tang et al. 2005].

Protocolo 4.1 `Thresh-Key-Gen(`$coord$`,` $\mathcal{V}$`)`

1: Cada $s_i \in \mathcal{V}$ define (2t+2) números aleatórios, $a_{im} \in GF(q)$ e $b_{im} \in GF(q)$ $(0 \leq m \leq t)$, como os coeficientes de dois polinômios de grau t: $f_i(z) = \sum_{m=0}^{t} a_{im} z^m$ e $f'_i(z) = \sum_{m=0}^{t} b_{im} z^m$
   1) s_i calcula: $k_{ij} = f_i(s_j)\ mod\ p$ e $k'_{ij} = f'_i(s_j)\ mod\ p$ $(\forall j \in \mathcal{V})$
   2) s_i calcula $(t+1)$ valores públicos: $P_{im} = (a_{im}T) \oplus (b_{im}T')(0 \leq m \leq t)$
2: s_i envia uma mensagem contendo k_{ij} e k'_{ij} para s_j $(j \in \mathcal{V})$, cifrada com a chave pública de s_j
3: s_i envia por *broadcast* uma mensagem contendo $\{P_{im} | 0 \leq m \leq t\}$
4: s_i, mediante o recebimento de k_{ji} e k'_{ji}, verifica: $(k_{ji}T) \oplus (k'_{ji}T') = \sum_{m=0}^{t\oplus} (k_i^m P_{jm})$
   se esta verificação falhar, s_i inclui s_j na sua lista negra e envia para o $coord$ e para os demais servidores uma acusação contra s_j contendo a mensagem enviada por s_j e a mensagem decifrada correspondente.
5: s_i também inclui um servidor s_l na sua lista negra se receber uma acusação válida contra o mesmo.
6: $coord$ cria o conjunto $\mathcal{Q}$ de servidores qualificados contendo todos os servidores em $\mathcal{V}$ que cumpriram o passo 3 e contra os quais não foi divulgada qualquer acusação válida.
7: $coord$ envia $\mathcal{Q}$ para todos os servidores em $\mathcal{V}$
8: s_i calcula $A_{i0} = a_{i0}T$ e envia A_{i0} para todos os demais servidores de $\mathcal{V}$.
9: s_i calcula a sua chave parcial k_i: $k_i = \sum_{j \in \mathcal{Q}} k_{ji}$.
10: s_i calcula a chave pública correspondente como: $y = \sum_{j \in \mathcal{Q}}^{\oplus} A_{j0}$

4.2. Assinatura baseada em Limiar

A geração de assinaturas é feita através do protocolo 4.2, que é baseado no protocolo MiDS-S proposto em [Tang et al. 2005], que por sua vez é a versão em curvas elípticas do algoritmo de assinatura Schnorr. A assinatura de uma mensagem msg é feita por

um grupo de servidores e cada um deles gera uma assinatura parcial da mensagem. A assinatura completa é resultado da combinação destas assinaturas parciais. A decisão sobre qual conjunto de assinaturas parciais usar cabe ao atual coordenador s_c.

Protocolo 4.2 `Thresh-Sig` $(msg, s_c, \mathcal{V}_{s_c})$

1: Cada $s_i \in \mathcal{V}_{s_c}$ executa `Thresh-Key-Gen` $(s_c, \mathcal{V}_{s_c})$ para gerar uma parte e_i de um segredo e que será usado apenas nesta assinatura (*one-time secret*), a correspondente chave pública $r = eT$, e uma chave parcial $r_i = e_i T$ derivada da chave pública.
2: s_i calcula $c = H(msg, \{r\}_x)$, sendo $\{r\}_x$ a coordenada x de r, e H uma função de *hash*.
3: s_i envia sig_i para os demais servidores $s_j \in \mathcal{V}_{s_c}$, onde $sig_i = e_i - c \prod_{l \neq i}(s_l(s_i - s_l)^{-1})k_i$
4: s_i recebe as assinaturas parciais e verifica se o seguinte conjunto de assinaturas parciais recebidas possui cardinalidade superior a t: $SIG_i = \{sig_j | (sig_j)T = r_j - cQ(j)y_j\}$ onde, $Q(j) = \prod_{t \neq j} s_t(s_j - s_t)^{-1}$
5: s_i gera a assinatura completa (c, sig) selecionando t assinaturas parciais de SIG_i e adiciona-as a sig_i.

4.3. Redistribuição de Chaves Parciais

Em essência , protocolos de redistribuição de chaves parciais [Desmedt and Jajodia 1997, Wong et al. 2002] consistem em cada membro de um subgrupo ω ($|\omega| \geq 2t + 1$) utilizar o esquema de Shamir [Shamir 1979] para dividir a sua chave parcial considerando a configuração (t', n') do grupo que receberá as novas chaves parciais. Cada membro deste novo grupo calcula a sua chave parcial por meio de um conjunto de partes recebidas dos membros que conduzem o processo. A redistribuição na ACD é feita pelo protocolo 4.3.

Protocolo 4.3 `Thresh-Redist` $(t', \omega, coord)$

1: Cada $s_i \in \omega$ define um polinômio $f_i(z) = k_i + \sum_{r=1}^{t'} c_{ir} z^r \ (mod\ p)$:
2: s_i calcula: $\widehat{k}_{ij} = f_i(j)\ mod\ p$ ($j \in \omega$); e os valores públicos $e_{ir} = g^{c_{ir}}\ mod\ p$, ($r \in \{0, ..., t'\}$).
3: s_i envia uma mensagem contendo $\widehat{k}_{ij}$ para s_j ($j \in \omega$), cifrada com a chave pública de s_j.
4: s_i envia por *broadcast* uma mensagem contendo $\{e_{ir} | 0 \leq r \leq t'\}$
5: s_i, mediante o recebimento de $\widehat{k}_{ji}$, verifica se: (i) $g^{\widehat{k}_{ji}} \equiv g^{k_j} \prod_{r=1}^{t'} (e_{jr})^{i^r}$
se esta verificação falhar, s_i inclui s_j na sua lista negra e envia para o $coord$ e para os demais servidores uma acusação contra s_j contendo a mensagem enviada por s_j e a mensagem decifrada correspondente.
6: s_i também inclui um servidor s_l na sua lista negra se receber uma acusação válida contra o mesmo.
7: $coord$ cria o conjunto $\mathcal{Q}$ de servidores qualificados contendo todos os servidores em ω que cumpriram o passo 4 e contra os quais não foi divulgada qualquer acusação válida.
8: $coord$ envia $\mathcal{Q}$ para todos os servidores em ω.
9: s_i calcula a sua nova chave parcial: $k'_i = \sum_{j \in \mathcal{Q}} \widehat{k}_{ji} b_j$ onde $b_j = \prod_{l \in \mathcal{Q}, l \neq j} \frac{l}{l - j}$

5. Considerações sobre a ACD e seu suporte algorítmico

Em relação aos serviços de certificação tradicionais de emissão, revogação e consulta de certificados, a base algorítmica da ACD contempla apenas a emissão de certificados. Os outros dois serviços não são tratados neste trabalho por exigirem mecanismos específicos que os viabilizem de forma eficiente em MANETs e as suas abordagens não são possíveis dentro do limite de espaço colocado para o artigo.

O uso de coordenadores, em particular de uma TPC, na execução dos protocolos criptográficos apresentados na seção 4 representa uma centralização, mesmo que parcial, nas operações da ACD, o que fere as características de distribuição e ausência de infra-estrutura fixa inerentes às MANETs. Entretanto, a alternativa ao uso de coordenadores consiste na adoção de protocolos de *broadcast* confiável e de consenso, os quais não seriam adequados por resultarem em um alto custo de comunicação. Tal custo foi o aspecto que nos fez decidir por esta centralização parcial.

Durante o funcionamento da ACD, a mobilidade constante dos nós (servidores ou não) possibilita a ocorrência de particionamentos da rede e eventualmente no grupo $\mathcal{S}$ de servidores. Estes isolamentos, às vezes temporários, durante uma *época*, não provocam ameaças à segurança e funcionamento correto do grupo $\mathcal{S}$. No modelo proposto neste trabalho, entretanto, pode ocorrer a ação maliciosa de nós servidores levando à partição da ACD durante as definições de uma nova *época*, compreendidas em um período de sincronização. O particionamento de $\mathcal{S}$ ocorre quando não há rotas ligando dois ou mais subgrupos de servidores da ACD, e poderia resultar da ação de servidores maliciosos (t servidores) que, em conluio ou isoladamente, participam da fase de sincronização tentando isolar nós corretos. Para que a coexistência de duas ou mais ACDs não ocorra na rede durante uma *época*, algumas medidas são necessárias.

Uma das medidas que adotamos na nossa proposta é usar na busca em largura apresentada na seção 3.2, o protocolo de roteamento sob demanda para redes ad hoc móveis apresentado em [Obelheiro and Fraga 2006]. O *flood* de uma mensagem `discovery-req` é enviado para todos os vizinhos do solicitante, e a medida que as respostas de servidores não atingem o valor mínimo necessário para a conexão do nó na rede, outras mensagens são enviadas considerando um alcance maior em número de *hops*. Quando um nó origem recebe o `discovery-rep`, o mesmo extrai a rota acumulada até o servidor destino e adiciona esta rota ao seu *cache*, inclusive se a rota conseguida for disjunta (duas rotas são disjuntas quando não compartilham nós intermediários) das demais rotas já armazenadas para o mesmo destino. O protocolo usado caracteriza um roteamento de origem (*source routing*). Ou seja, o nó de origem no envio de pacotes escolhe rotas para um destino que não são faltosas e procura alternativas no *cache* quando ocorre falhas no envio de pacotes a um determinado destino. Rotas alternativas e disjuntas minimizam a ação de t nós maliciosos que tentam forçar isolamentos durante o período de sincronização. Nós cujas visões não possuem o mínimo de servidores estabelecido, devem executar o procedimento de entrada na rede novamente.

A outra medida tomada é definir o número de servidores em: $4t{+}2 \geq |\mathcal{S}| \geq 2t{+}1$. Estes limites devem impedir que se forme duas ACDs em uma *época*. Se nós ficarem isolados e suas partições não atingirem o valor mínimo para caracterizar uma ACD, ficarão inativos (ou seja, na *época* antiga) e não vão conseguir interferir no funcionamento da partição majoritária que vai formar a ACD da nova *época*. Pode também ocorrer que nenhuma das partições atinja o valor mínimo na composição e, neste caso, a retomada do sistema só se dará com a interferência do administrador.

5.1. Custos de Comunicação

Os custos de comunicação dos algoritmos propostos possuem uma forte dependência dos custos dos protocolos criptográficos adotados. Na tabela 1 são mostrados os custos derivados de uma verificação dos algoritmos propostos (algoritmos 3.1, 3.2 e 3.3) em termos de mensagens transmitidas. Estes custos foram obtidos tomando como base a análise feita em [Tang et al. 2005]. Sobre os dados de transmissão apresentados na tabela, os mesmos são mostrados no formato (x, y), onde x indica o custo de transmissão em canais *unicast* e y representa *broadcasts*.

Na tabela 1, a segunda coluna apresenta os custos por servidor e a coluna subseqüente apresenta um pequeno exemplo. Com base neste exemplo, tendo a ACD formada por 10 servidores e tolerando 2 servidores maliciosos, o comportamento geral do sistema no estabelecimento da *época* 0, envolverá um total de 270 mensagens enviadas

Table 1. Custos de Comunicação dos Algoritmos da ACD.

Algoritmo	**Transmissão** (*por servidor*)	**Exemplo (**$n = 10, t = 2$**)**
`ACD-Initialization`	$(3n-3,\ 2n+6t+7)$	$(27,\ 39)$
`Issue-Certificate`	$(2n,\ n+2t+2)$	$(20,\ 16)$
`Synchronization`	$(3n,\ n+3t+3)$	$(30,\ 19)$

por *unicast* e 390 *broadcasts*. Neste mesmo exemplo, a emissão de um novo certificado envolveria 200 *unicasts* e 160 *broadcasts*. Por sua vez, a sincronização da ACD durante a transição para uma nova *época* envolveria 300 *unicasts* e 190 *broadcasts*.

Embora uma análise com relação à eficiência assintótica dos algoritmos seja possível, oferecendo uma caracterização mais simples do custo, a mesma não seria adequada ao contexto deste trabalho, uma vez que uma ACD envolve normalmente uma composição com um número reduzido de servidores ($n = 10$, por exemplo). Assim, neste caso, a eficiência assintótica auferida não seria efetiva em situações reais.

6. Trabalhos Relacionados

A literatura apresenta trabalhos que propõem autoridades certificadoras distribuídas para MANETs [Luo et al. 2005, Yi and Kravets 2003, Zouridaki et al. 2004], os quais fundamentam-se em técnicas de criptografia de limiar para gerenciar a chave privada da AC. Entretanto, estas propostas não apresentam mecanismos eficientes que tratem do aspecto dinâmico da arquitetura da AC, e assumem que os parâmetros t e n permanecem fixos durante todo o ciclo de vida do sistema.

Zhou e Haas [Zhou and Haas 1999] introduziram o problema de redistribuir as partes da chave privada de uma AC distribuída para MANETs em função das mudanças ocorridas em um grupo dinâmico de servidores. Como solução eles descrevem a técnica de redistribuição de partes de um segredo para uma nova estrutura de acesso [Desmedt and Jajodia 1997], não apresentando no entanto um mecanismo que viabilize na prática a reconfiguração da AC. De maneira similar, em [Narasimha et al. 2003] é explorado o uso de criptografia de limiar para controle de *membership* em sistemas P2P, permitindo que nós que entram no sistema sejam dinamicamente configurados com um certificado de grupo e com uma parte da chave privada do grupo. Entretanto, este trabalho limita-se a apresentar soluções criptográficas que viabilizam a emissão distribuída deste certificado e da nova parte da chave privada. No que refere-se à auto-adaptação do sistema em virtude das mudanças na composição do grupo, o trabalho apenas discute o problema e aponta direções para solucioná-lo.

Em uma proposta mais recente, Wu *et al* [Wu et al. 2005] apresentam um AC para MANETs capaz de tratar a entrada de novos servidores através de um protocolo que usa um subgrupo de servidores para criar uma nova parte da chave privada da AC e entregá-la ao novo servidor. Porém, a arquitetura do serviço não mantém uma proporção segura entre o limiar t e o total de servidores n, uma vez que são criadas partes da chave privada da AC indefinidamente, sem que o limiar t seja redefinido.

7. Conclusões

Neste trabalho discutimos as conseqüências decorrentes do *churn* em uma autoridade certificadora que opera em uma MANET, e introduzimos uma base algorítmica que permite a esta AC se auto-adaptar aos eventos constantes de entrada e saída dos dispositivos computacionais móveis sobre os quais são mantidas as suas funcionalidades. No projeto dos

algoritmos foram estabelecidas premissas com base em modelos de sistemas dinâmicos de modo a assegurar as terminações dos algoritmos propostos. A execução das funções de auto-adaptação e gerenciamento da AC foram agrupadas em períodos específicos de sincronização a fim de reduzir o impacto e os custos gerados pelo uso de protocolos de criptografia de limiar usados na configuração de novos servidores e na reconfiguração dos elementos de segurança em vigor no sistema.

References

Aguilera, M. K. (2004). A pleasant stroll through the land of infinitely many creatures. *ACM SIGACT News*, 35(2):36–59.

Desmedt, Y. and Jajodia, S. (1997). Redistributing secret shares to new access structures and its applications. Technical Report ISSE TR-97-01, George Mason University.

Godfrey, P. B., Shenker, S., and Stoica, I. (2006). Minimizing churn in distributed systems. *SIGCOMM Comput. Commun. Rev.*, 36(4):147–158.

Koblitz, N., Menezes, A., and Vanstone, S. (2000). The state of elliptic curve cryptography. *Designs, Codes and Cryptography*, 19(2-3):173–193.

Luo, J., Hubaux, J.-P., and Eugster, P. T. (2005). Dictate: Distributed certification authority with probabilistic freshness for ad hoc networks. *IEEE Transactions on Dependable and Secure Computing*, 02(4):311–323.

Mostefaqui, A., Raynal, M., Travers, C., Patterson, S., Agrawal, D., and Abbadi, A. E. (2005). From static distributed systems to dynamic systems. In *Reliable Distributed Systems, 2005. SRDS 2005. 24th IEEE Symposium on*, pages 109– 118.

Narasimha, M., Tsudik, G., and Yi, J. H. (2003). On the utility of distributed cryptography in p2p and manets: the case of membership control. In *11th IEEE International Conference on Network Protocols (ICNP'03)*.

Obelheiro, R. R. and Fraga, J. S. (2006). A lightweight intrusion-tolerant overlay network. *isorc*, 0:496–503.

Pedersen, T. P. (1991). A threshold cryptosystem without a trusted party. *Advances in Cryptology - EUROCRYPT'91*, 547:522–526.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

Tang, C., Chronopoulos, A. T., and Raghavendra, C. S. (2005). Soft-timeout distributed key generation for digital signature based on elliptic curve d-log for low-power devices. In *Security and Privacy for Emerging Areas in Communications Networks, 2005.*

Wong, T. M., Wang, C., and Wing, J. M. (2002). Verifiable secret redistribution for threshold sharing schemes. Technical Report CMU-CS-02-114, Carnegie Mellon University.

Wu, B., Wu, J., Fernandez, E. B., Ilyas, M., and Magliveras, S. (2005). Secure and efficient key management in mobile ad hoc networks. *Journal of Net.and Comp.Applic.*

Yi, S. and Kravets, R. (2003). Moca: Mobile certificate authority for wireless ad hoc networks. In *2nd Annual PKI Research Workshop*.

Zhou, L. and Haas, Z. J. (1999). Securing ad hoc networks. *IEEE Network*, 13(6):24–30.

Zouridaki, C., Mark, B. L., Gaj, K., and Thomas, R. K. (2004). Distributed ca-based pki for mobile ad hoc networks using elliptic curve cryptography. In *The 1st European PKI Workshop on Research and Applications (EuroPKI 2004)*, pages 232–245.