无线传感器网络对偶密钥误用检测算法①

苏 忠, 徐 洸, 丛培荣

(空军指挥学院 网络中心, 北京 100097)

摘 要: 无线传感器网络正常节点之间的对偶密钥有可能受损, 攻击者将误用受损密钥伪造信息数据包, 破坏正常通信或消耗节点有限的资源. 针对这一问题, 提出一种对偶密钥误用检测算法. 在每一个信息数据包里附加一个可验证的认证因子, 转发节点通过验证认证因子就能够确认对偶密钥是否被误用. 通过安全分析和性能分析表明该检测算法的有效性. 而且, 该算法可集成到大多的密钥预分发算法或错误数据过滤算法, 以提供更可靠的安全通信.

关键词: 误用对偶密钥; 认证因子; Bloom Filter; 无线传感器网络; 安全

Algorithm for Misused Pairwise Keys Detection in Wireless Sensor Networks

SU Zhong, XU Guang, CONG Pei-Rong

(Network Center, Air-Force Command College, Beijing 100097, China)

Abstract: In wireless sensor networks, the pairwise keys shared by non-compromised nodes could be compromised, then the adversary would misuse the pairwise keys to forge message report in order to destroy secure communication or deplete the limited resource in sensor nodes. To address this issue, in this paper, an algorithm is proposed to detect misused pairwise keys. Specially, an efficient verifiable authenticator is attached to each message report. The forwarding node can detect whether the pairwise key is misused by verifying the authenticator. The security analysis and performance analysis demonstrate the efficiency of the proposed algorithm. Moreover, the proposed algorithm can be integrated with most of key pre-distribution schemes or false data filtering schemes to provide more reliable secure communication.

Key words: misused pairwise key; authenticator ; Bloom Filter; wireless sensor networks; security

在某些无线传感器网络应用环境里, 攻击者并不愿意公开暴露自己的身份, 否则将冒被识别出来的风险. 攻击者更喜欢假冒正常节点, 发送伪造数据包, 干扰和破坏网络的正常运行. 要做到这一点, 获取正常节点的对偶密钥是最重要的一步. 如果正常节点的对偶密钥被攻击者误用(misuse)而没有采取相应的检测机制去对误用的对偶密钥进行检测, 伪造的数据包则无法被识别, 从而严重干扰和破坏网络功能的正常运行[1]. 而且, 由于传感器节点的资源本身就非常受限, 若节点资源被蓄意耗费, 将缩短节点的工作寿命, 这就要求所有的伪造数据包越早被识别出来越好. 因此, 有效的对偶密钥误用检测对于保证无线传感器的安全通信具有重要意义, 值得深入研究.

在无线传感器网络应用环境里, 节点之间的对偶密钥的受损可能来自两个方面: 一是节点受损, 其所使用的对偶密钥必定受损; 二是攻击者根据所获取的受损节点信息而推导获得正常节点的对偶密钥. 对于第一种情况, 一般通过节点的受损检测机制来确定; 对于第二种情况, 根据无线传感器网络的特点及当前密钥管理研究进展, 基于对称密钥管理的算法都不同程度地存在着这个问题. 但是, 当前专门针对第二种情况的节点对偶密钥误用检测研究成果并不多见.

① 基金项目:国家自然科学基金(61071065)
收稿时间:2013-12-31;收到修改稿时间:2014-03-07

本文提出一种对偶密钥的误用检测算法. 该算法的主要设计思想是在每一个传送的数据包里增加一个认证因子, 转发(接收)节点通过重构认证因子并与数据包里的认证因子比较, 若攻击者使用对偶密钥来伪造数据包, 对偶密钥会立即被检测为误用, 因为攻击者无法伪造正确的认证因子. 伪造的数据包也被立即丢弃.

1 相关研究工作

Liu 等人较早关注误用对偶密钥检测问题[2], 并提出集中式检测算法和分布式检测算法(以下统称 “Liu 算法”). 在集中式检测算法里, 所有的检测数据包都发送给基站, 由基站验证数据包从而发现对应节点的对偶密钥是否被误用. 这种检测算法虽然检测精度高, 但存在着节点通信开销却会随着检测频率的增多而急剧增大, 影响节点正常通信的缺点; 在分布式检测算法里, 由指定的检测节点进行对偶密钥误用检测, 在一定程度上克服了集中式检测算法的缺点, 但由于检测节点有可能被俘获, 或者工作状态不正常, 无法保证检测的准确率. 而且, 不管是借助基站还是特定节点进行误用对偶密钥检测, 都要生成一个检测数据包并多跳发送到指定节点, 因此会产生检测延迟, 从而很容易受到 DoS 攻击, 算法的容侵性较差.

Kim 等人使用概率检测方法[3], Park 等人使用节点分簇方法[4], Kang 等人使用标记(Token)方法[5], Han 等人使用双向检测方法[6], Goyal 等人使用动态密钥方法[7]对 Liu 算法进行改进, 在降低通信开销带来的能耗方面取得进展. 但在检测延迟所导致 DoS 攻击问题所取得的效果并不明显.

与上述算法相比, 本文所提出的算法的主要创新有两点: 一是任何被误用的对偶密钥都会在一跳之内被正常节点检测出来, 这样能够很大程度节约通信开销和计算开销, 并且比较适用于大规模的多跳网络; 对偶密钥的误用检测不需要特殊的需求, 如预先确定的节点部署信息、时间同步、额外的检测节点或基站的协助等; 二是误用检测的准确率与节点的受损数量无关, 不管有多少数量的受损节点, 都不影响正常节点对对偶密钥的误用检测, 具有较强的容侵特性.

2 预备知识

在描述所提算法之前, 首先对该算法所需使用到的一些知识进行简要介绍, 包括椭圆曲线密码体制和 Bloom Filter[8].

2.1 椭圆曲线密码体制

研究表明[9,10]: 由于椭圆曲线密码体制具体较小的密钥长度, 更快的计算速度, 以及对存储和能量的需求比较适中, 比较适用于无线传感器网络. 本算法使用标准的椭圆曲线 Diffie-Hellmann(ECDH)密钥交换算法[11]来实现在认证密钥阶段的信息交换.

2.2 Bloom Filter

众所周知, 在所有的公钥密码体制里, 除 IBC 之外, 要使用其他节点的公钥时, 必须对公钥进行认证, 也就是确认该公钥是否真的属于特定的对象, 否则任意对象都可声称拥有某一个公钥, 从而无法保证信息的机密性. 本算法将使用 Bloom Filter 技术来实现对公钥的认证.

Bloom Filter 是一个长度为 m 比特的比特向量, 比特向量的每一位初始值都设置为 0. 假定有 n 个对象 $\{O_1,O_2,...,O_n\}(n \square m)$, 对于每一个给定对象 $O_i(1 \le i \le n)$, 使用 k 个单向哈希函数 H_{BF}^1, H_{BF}^2, …, H_{BF}^k, 计算得到 $H_{BF}^1(O_i)$, $H_{BF}^2(O_i)$ …, $H_{BF}^k(O_i)$. 这些哈希值的范围为从 1 到 m. 若 $H_{BF}^l(O_i)(1 \le l \le k)$ 的值为 b, 则把对应的 Bloom Filter 的第 b 比特位设置为 “1”. 由于有些哈希值可能相等, 因此在 Bloom Filter 里某些比特位可能被重复设置为“1”.

若要验证某一个对象 $O_j(1 \le j \le n)$, 则检验 Bloom Filter 对应的 $H_{BF}^1(O_j)$, $H_{BF}^2(O_j)$, …, $H_{BF}^k(O_j)$ 比特位是否全部为“1”, 若有任意一个对应的比特位为零, 则对象 O_j 不能通过验证. 反之, 若对应的 k 个比特位全部为“1”, 则该对象通过验证.

在 Bloom Filter 里, 一定不存在漏报(false negative), 即任何属于 Bloom Filter 的对象一定会被检测出来; 但可能存在着误报 (false positive), 即被验证通过的对象不一定属于 Bloom Filter. 因此在使用 Bloom Filter 时, 应考虑在满足应用需求的前提下如何最大限度降低误报率.

3 算法描述

本算法共包括四个阶段, 分别为: 节点初始化阶段、认证密钥交换阶段、对偶密钥检测阶段和误用报告检测阶段, 如图 1 所示.

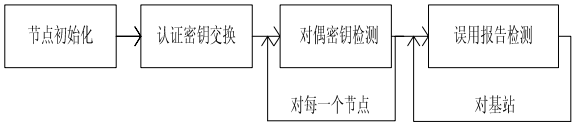

图 1　算法的四个阶段

其中, 前两个阶段, 即节点初始化、认证密钥交换仅执行一次, 而后两个阶段根据实际需求可能反复执行. 对于每一个节点而言, 当接收到上一跳节点发送过来的数据包, 就启动对偶密钥的误用检测; 对于基站而言, 当接收到节点发送过来的误用报告时, 就启动误用报告检测.

设网络的节点总数为 N. 下面对各个阶段所需完成的工作进行详细描述. 在此之前, 首先给出两个定义:

定义 1.　认证密钥(Authentication Key). 任意一个节点 u_i, 对另一个节点 u_j 的认证密钥(Authentication Key)可表示为 $AuthKey_{u_i}^{u_j}$, 该认证密钥是节点 u_i 的私钥 SK_{u_i} 和 u_j 的标识 ID_{u_j} 的并置哈希值, 如式(1)所示.

$$AuthKey_{u_i}^{u_j}=H\left(SK_{u_i}\| ID_{u_j}\right) \tag{1}$$

在式(1)里, $1\le i,j\le N$ 且 $i\ne j$, $H(\ )$ 是一个安全的单向哈希函数. $AuthKey_{u_i}^{u_j}$ 主要用于数据包的认证, 也就是, 当节点 u_i 接收到声称来自 u_j 的数据包时, 就使用该认证密钥来生成认证因子并认证数据包源的正确性.

定义 2　认证因子(Authenticator). 节点 u_i 对于另一个节点 u_j 的认证因子(Authenticator) $Authenticator_{u_i}^{u_j}$, 表示为被传送的信息 M 的哈希值、认证密钥 $AuthKey_{u_i}^{u_j}$ 和最新的计数器 IV_{u_i,u_j} 的并置哈希值, 如式(2)所示.

$$Authenticator_{u_i}^{u_j}=H\left(H(M)\left\|AuthKey_{u_i}^{u_j}\right\|IV_{u_i,u_j}\right) \tag{2}$$

使用认证因子 $Authenticator_{u_i}^{u_j}$, 可以保证当前发送或转发数据包的节点以及下一跳节点的正确性. 任何伪造当前节点及下一跳节点的数据包都会因为无法伪造正确的认证因子而被检测出来.

3.1 节点初始化

在节点部署之前, 可信的离线服务器给每一个节点 $u_i\left(1\le i\le N\right)$ 预分发一个私钥 SK_{u_i}, 并同时生成一个公钥 $PK_{u_i}=SK_{u_i}\times G$ 并预分发给 u_i, 这里 G 是椭圆曲线上的一个固定点. 每一个公钥可以公开, 为其他所有节点知道, 但私钥仅仅为节点本身及基站所知道.

为了对公钥进行认证, 每一个节点还必须预分发一个 Bloom Filter. Bloom Filter 的构造过程如下: 离线服务器使用所有节点的公钥及对应节点的标识生成一个数据集 S, S 可表示如式(3)所示:

$$S=\left\{\left(PK_{u_i}\left\|ID_{u_i}\right.\right)\middle|1\le i\le N\right\} \tag{3}$$

然后, 离线服务器使用 k 个哈希函数, 把数据集 S 里的每一个元素映射到一个长度为 m 比特的数据向量 $a_0a_1\ldots a_{m-1}$ 里. 每一个 $a_l\left(0\le l\le m-1\right)$, 可表示如式(4)所示:

$$a_l=\begin{cases}1\ if\exists i,t,i\in[1,N],\ t\in[1,k]s.t.H_{BF}^{t}\left(PK_{u_i}\left\|ID_{u_j}\right.\right)=1\\ 0\ otherwise\end{cases} \tag{4}$$

上述的操作如图 2 所示. 当用于表示 Bloom Filter 的数据向量 $a_0a_1\ldots a_{m-1}$ 构造完毕, 每一个节点都存储该 Bloom Filter, 同时保存 k 个哈希函数.

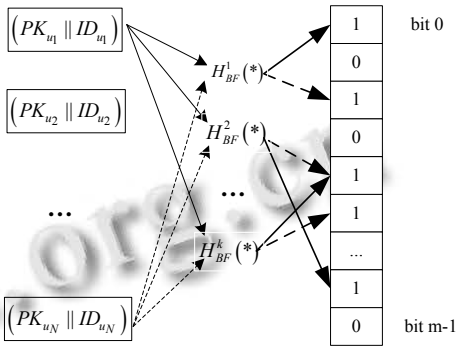

图 2　每一个节点存储的 Bloom Filter

预分发以上数据信息后, 节点就可以部署到指定的区域里. 在这里假定: 每两个节点要进行通信时, 都能够通过某一个密钥预分发方案或协议建立它们之间的对偶密钥. 至于使用哪一个密钥预分发算法或协议, 在本算法里并不特别强调. 此外, 基站预分发所有节点的公/私钥.

3.2 认证密钥交换

部署完成后, 相邻节点需要进行认证密钥交换. 当两个相邻节点 u_i 和 u_j 要交换它们之间的认证密钥时, 它们可执行如下步骤(假定 u_i 首先发出请求):

步骤 1　u_i 发送一个 HELLO 数据包给 u_j, 该数据

包包含如下信息: u_i 的公钥 $\mathrm{PK}_{\mathrm{u_i}}$, 标识 $ID_{\mathrm{u_j}}$ 以及初始的随机计数器 IV_{u_i,u_j}:

$$u_i \to u_j : \mathrm{PK}_{\mathrm{u_i}}, ID_{\mathrm{u_j}}, IV_{u_i,u_j}$$

步骤 2 当 u_j 接收该 HELLO 数据包, 首先对公钥进行认证, 也就是确认公钥及节点标识是否属于数据集 S 的一个元素. 为了做到这一点, u_j 使用预分发的 k 个哈希函数计算 $H_{BF}^{t}=\left(PK_{u_i}\middle\|ID_{u_i}\right), t\in[1,k]$, 然后与预分发的 Bloom Filter 相对应的比特位进行比较, 如果对应比特位全部为 “1”, 则公钥通过认证, 否则, 只要有一个对应的比特位为 “0”, 公钥就无法通过认证, u_j 将丢弃该 HELLO 数据包, 中断此阶段与 u_i 的后续操作.

步骤 3 若公钥通过认证, u_j 计算一个通信密钥 $\mathrm{SK}_{\mathrm{u_j}}\times\mathrm{PK}_{\mathrm{u_i}}$, 使用该通信密钥加密哈希值 $H\left(SK_{u_j}\middle\|ID_{u_i}\right)$, 向 u_i 回复一个数据包, 该数据包包括: 一个加密的哈希值、u_j 的公钥及其标识:

$$u_i \leftarrow u_j : \mathrm{PK}_{\mathrm{u_j}}, ID_{\mathrm{u_i}}, E\left(SK_{u_j}\times PK_{u_i}, H\left(SK_{u_j}\middle\|ID_{\mathrm{u_i}}\right)\right)$$

此外, 还设置自己的随机计数器 $IV_{u_j,u_i}=IV_{u_i,u_j}$, 这样确保两个节点的计数器同步. $E(k,M)$ 表示使用密钥 k 对信息 M 进行加密.

步骤 4 接收到 u_j 回复的数据包, u_i 首先认证的公钥, 方法如同步骤 2 所述. 若公钥认证通过, u_i 将计算另一个通信密钥 $SK_{u_i}\times PK_{u_j}$, 根据 ECDH 算法的特性, 有:

$$SK_{u_i}\times PK_{u_j}=SK_{u_i}\times\left(SK_{u_j}\times G\right)=SK_{u_j}\times\left(SK_{u_i}\times G\right)$$
$$=SK_{u_j}\times PK_{u_i}$$

也就是说, 两个通信密钥是相同的, 这就意味着 u_i 能够自己计算的通信密钥解密所接收的加密哈希值, 从而获得认证密钥 $AuthKey_{u_j}^{u_i}$.

步骤 5 按同样的方法, u_i 计算哈希值 $H\left(SK_{u_i}\middle\|ID_{u_j}\right)$, 使用对偶密钥加密该哈希值, 连同自己的公钥及并发送给 u_j. 这样, u_j 也可获得认证密钥 $AuthKey_{u_i}^{u_j}$.

每一对相邻节点仅执行一次该阶段的操作. 不是相邻节点无须进行认证密钥交换.

3.3 对偶密钥误用检测

当节点 u_i 要发送或转发数据包给下一跳节点 u_j 时, 它首先根据式(2)构造所需的认证因子. 如果信息 M 是新发送的, u_i 把认证因子加入到数据包里, 否则, 若是转发数据包, u_i 则用新的认证因子替代旧的认证因子.

从 u_i 发送到 u_j 的数据包可表示如下:

$$u_i \to u_j : ID_{\mathrm{u_i}}, E(K_{u_i,u_j}, M, IV_{u_i,u_j})\left|Authenticator_{u_i}^{u_j}\right.$$

其中, K_{u_i,u_j} 为两个节点之间已建立的对偶密钥. 计数器 IV_{u_i,u_j} 是一个单调上升的数值, 能够确保在节点的运行期间其值不重复.

当 u_j 接收到该数据包, 按以下步骤对对偶密钥 K_{u_i,u_j} 进行检测:

步骤 1 首先使用对偶密钥解密该数据包, 若无法解密, 则丢弃该数据包; 否则检查随机计数器是否最新, 若不是, 也丢弃该数据包, 否则转到步骤 2.

步骤 2 使用认证密钥 $Authenticator_{u_i}^{u_j}$、信息 M 和随机计数器 IV_{u_i,u_j} 计算出对应该数据包的认证因子, 然后与附加在数据包里的认证因子比较, 若不相同, 转到步骤 3, 否则转到步骤 4.

步骤 3 对偶密钥 K_{u_i,u_j} 被误用, 该数据包被立即丢弃. 而且, 构造一个对偶密钥误用报告发送给基站 B, 该报告包括两个认证密钥 $AuthKey_{u_i}^{u_j}$ 和 $AuthKey_{u_j}^{u_i}$. 误用报告的格式如下所示:

$$u_i \to B : ID_{\mathrm{u_i}}, ID_{\mathrm{u_j}}, H\left(AuthKey_{u_i}^{u_j}\middle\|AuthKey_{u_j}^{u_i}\right)$$

步骤 4 对偶密钥未被误用 u_j 确定下一跳节点 u_{r} 后, 使用认证密钥 $AuthKey_{u_i}^{u_r}$, 随机计数器 IV_{u_i,u_r} 以及信息 M 构造新的认证因子, 并取代数据包里旧的认证因子, 同时也使用新的随机计数器替换旧的随机计数器. 然后把数据包转发给 u_{r}.

3.4 误用报告验证

当基站接收到对偶密钥误用报告, 由于它拥有所有节点的公钥与私钥, 所以它能够检测该报告的真实性. 具体做法如下: 它根据报告里的节点标识, 构造对应的认证密钥, 并计算它们的并置哈希值, 与报告的哈希值比较, 若相等, 则报告为真实的, 否则为伪造的. 若报告的真实性得到验证, 基站知道对应的两个节点之间的对偶密钥已受损, 便可采取相应措施处理受损的对偶密钥. 若报告的真实性得不到验证, 基站丢弃该数据包, 不采取其他任何处理措施.

4 安全分析

本算法能够提供有效的防范措施, 对付无线传感器网络里典型的攻击. 现分别分析如下.

4.1 假冒攻击

如果攻击者假冒正常的节点 u_i, 发送数据包给下一跳节点 u_j, 要想让该数据包通过认证, 攻击者必须获取认证密钥 $AuthKey_{u_i}^{u_j}$ 或 u_i 的私钥 SK_{u_i}. 然而, u_i 在传送数据包过程中从不暴露自己的私钥; 对于认证密钥 $AuthKey_{u_i}^{u_j}$, 它存在于数据包的认证因子里, 尽管攻击者获取对偶密钥就可以获得当前的认证因子, 但是, 获得当前认证因子并不意味着能够获取认证密钥. 根据式(2), 由于哈希函数的单向属性, 攻击者无法通过认证因子来计算认证密钥. 因此, 只要两个节点 u_i 和 u_j 是正常(未受损)的, 即使它们的对偶密钥已暴露, 攻击者也无法获取认证密钥或节点的私钥.

如果攻击者俘获了节点 u_j, 从而获取认证密钥 $AuthKey_{u_i}^{u_j}$, 但他也无法假冒正常节点 u_i 向其他节点发送数据包. 这是因为: $AuthKey_{u_i}^{u_j}$ 仅仅为 u_i 和 u_j 所使用. 攻击者无法生成为其他节点所使用的认证因子.

因此, 攻击者无法假冒任何正常的节点来发起假冒攻击. 换句话说, 如果两个正常节点的对偶密钥受损, 该受损密钥将立即被其中的一个正常节点检测出来.

考虑节点受损的情况. 一旦节点受损, 其所存储的秘密信息将完全暴露给攻击者, 这些已暴露的秘密信息对其他节点的安全造成威胁. 然而, 在本算法里, 每一个正常节点都能够独立地进行对偶密钥误用检测, 无须其他任何节点的协助. 因此, 不管受损节点的数量有多少, 都不影响正常节点的检测的正确性. 也就是说, 任意两个正常节点之间的对偶密钥被误用, 都会被其中一个正常节点检测出来.

需指出的是, 本算法致力于解决正常节点的对偶密钥的误用检测的问题. 如果节点已受损, 与它相邻的节点的对偶密钥当然已受损, 这一类问题不在本算法的讨论范围之内. 目前, 有部分算法或协议[12-15]专门致力于研究节点的受损检测问题, 在这里不再一一详细阐述.

4.2 伪造攻击

攻击者直接伪造认证密钥 $\overline{AuthKey_{u_i}^{u_j}}$, 并发送给节点 u_j. 尽管攻击者能够获取 u_i 的公钥 PK_{u_i} 及标识 ID_{u_i} (这是通过公钥认证的必要条件！), 但他无法生成通信密钥 $SK_{u_i} \times PK_{u_j}$, 这样就导致 u_j 无法解密攻击者伪造的数据包, 伪造的数据包将被丢弃.

由于攻击者无法获取正常节点的私钥或认证密钥, 因此也无法伪造认证因子.

攻击者有可能伪造关于正常节点 u_i 和 u_j 的对偶密钥误用报告, 并把报告发送给基站, 企图让基站相信 u_i 和 u_j 之间的对偶密钥已受损. 但由于攻击者既无法伪造认证密钥 $AuthKey_{u_i}^{u_j}$, 也无法伪造另一个认证密钥 $AuthKey_{u_j}^{u_i}$, 因此不能构造正确的哈希值 $H\left(AuthKey_{u_i}^{u_j} \| AuthKey_{u_j}^{u_i}\right)$, 从而无法伪造正常节点之间的对偶密钥误用报告. 换句话说, 正常节点之间的对偶密钥若没有受损, 则永远不会被基站误认为被受损了.

4.3 重传攻击

每一个数据包里都包含了一个最新的随机计数器, 由于该随机计数器具有单调上升的特性, 因此, 若接收节点发现当前接收到的数据包的随机计数器小于其已有的计数器值, 则认为该数据包为重传, 此数据包被丢弃. 重传的数据包将在一跳之内被发现, 从而被阻止进入网络.

4.4 拒绝服务攻击

攻击者可能发送大量伪造的数据包, 迫使正常节点使用有限的存储空间缓存这些数据包并等待处理. 但在本算法里, 由于每一个数据包都被当前的接收节点进行对偶密钥检测, 无须使用缓存等待处理, 或转送给其他特殊的节点进行处理, 因此攻击者无法发起拒绝服务攻击来破坏本算法的有效性.

5 性能分析

在本节里, 我们主要讨论在达到最小的 Bloom Filter 误报概率的前提下, 可支持的网络规模问题, 以及节点的存储、计算和通信开销分析. 为了便于计算, 假定 WSN 的节点密度为 6, 即每一个节点的平均邻居节点数为 6.

5.1 可支持的最大网络节点

在本算法里, 节点预分发一个 Bloom Filter 用于对公钥的认证. 如前所述, Bloom Filter 存在着误报概率, 也就是, 对于某一个不属于数据集 S 的元素 s, 可能会存在所有的哈希值 $H_{BF}(s)$ 在 bloom Filter 相对应的比特位里其值均为 “1”, 从而被误认为是属于数据集 S 的一个元素. 误报概率的大小, 将直接影响公钥认证的正确性. 所以本算法的一个主要设计目标是必须使

得误报概率尽可能地小.

为此, 这里需回答一个问题: 误报概率是如何影响到网络节点总数的? 也就是: 在达到指定的误报概率的前提下, 网络可支持的规模可达什么程度? 我们使用以下定理来回答这个问题.

定理 1. 给定 Bloom Filter 的长度为 m(比特), 最小的误报概率为 $f_{\min}$, 则可支持的网络的最大节点总数 N 应满足

$$N \approx \frac{-m(\ln 2)^2}{\ln f_{\min}} \tag{5}$$

证明: 假设用于Bloom Filter的哈希函数的数量为 k, 根据文献[8]所述, 误报概率 f 可表示为

$$f = \left(1-\left(1-\frac{1}{m}\right)^{kN}\right)^k \approx (1-e^{\frac{-kN}{m}})^k$$

$$= e^{k\ln(1-e^{\frac{-kN}{m}})}$$

令 $g = k\ln(1-e^{\frac{-kN}{m}})$, 最小化误报概率 f 相当于关于 k 最小化 g, 因此我们可得到

$$\frac{\partial g}{\partial k} = \ln(1-e^{\frac{-kN}{m}}) + \frac{kN}{m}\times\frac{e^{\frac{-kN}{m}}}{1-e^{\frac{-kN}{m}}}$$

当 $k = \frac{m}{N}\ln 2$ 时, 左边导数结果为 0. 因此, 最小的误报概率应为:

$$f_{\min} \approx (1-e^{\frac{-kN}{m}})^k = \left(1-e^{\frac{m\ln 2}{N}\times\frac{N}{m}}\right)^{\frac{m\ln 2}{N}}$$

$$= \left(2^{-\ln 2}\right)^{\frac{m}{N}}$$

由上式可推导出

$$N \approx \frac{-m(\ln 2)^2}{\ln f_{\min}}$$

证毕.

图 3 显示了在不同的最小误报概率的情况下, 可支持的最大节点数与 Bloom Filter 长度之间的关系.

从图 3 可以看出, 当最小误报概率值固定时, Bloom Filter 的长度越长, 可支持的最大节点数就越大; 当固定 Bloom Filter 的长度时, 最小概率越大, 可支持的最大节点数就越大. 因此, 可通过提高 Bloom Filter 的长度, 或降低最小误报概率, 来提高可支持的最大节点数. 例如, 如图 3 所示, 当 Bloom Filter 的大小为 20KB 时, 最小误报概率为 5×10^{-4} 可支持大约 10000 个节点, 而最小误报概率为 5×10^{-3} 则可支持大约 15000 个节点.

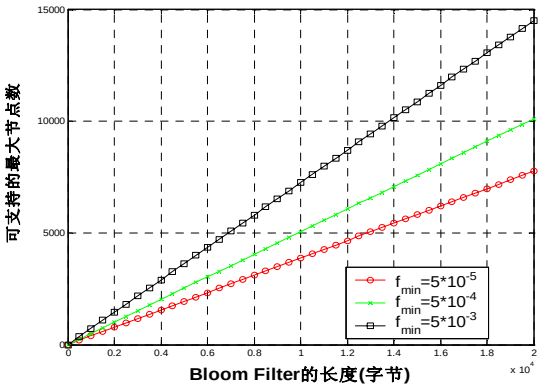

图 3 最大节点数与 Bloom Filter 长度

从上面的分析可以看出, 通过增长 Bloom Filter 的长度, 我们就可以在较小的误报概率下支持较大的网络规模.

5.2 存储开销分析

在本算法里, 每一个节点存储以下信息:

首先, 节点存储一个私钥和一个公钥以及节点标识. 私钥大小为 20 字节, 公钥大小为 40 字节, 而节点标识为 2 字节(假定网络节点数不超过 64K). 因此, 一共占用 62 字节.

其次, 节点存储一个长度为 m 的 Bloom Filter, 如上述分析, 长度 m 的大小取决于网络规模以及所需达到的误报率. 由于每一个节点预分发的 Bloom Filter 的内容都是固定的, 因此可把 Bloom Filter 存储在节点的 ROM 里.

第三, 每一个节点存储与其通信的节点的认证密钥. 每一个节点则平均需存储 6 个认证密钥和 6 个随机计数器, 若使用 RC5[10]实现哈希操作, 则每一个哈希输出值为 8 字节, 假设随机计数器的长度为 3 字节, 则节点需 66 字节存储认证密钥和随机计数器.

因此, 在本算法里, 每个节点需要使用 126 字节存储公钥、私钥、节点标识、认证密钥和随机计数器等信息, 另外, 占用一定的 ROM 用于存储 Bloom Filter(视网络需支持的最大节点数及最小误报率而定).

5.3 通信开销分析

实验表明[17]: 在 Crossbow MICA2DOT 里使用的 Chipcon CC1000 无线电模块在数据率为 12.4Kb/s 的情况下, 接收或传送一个字节需要消耗 28.6μJ 或 59.2μJ

的能量.

在认证密钥交换阶段里, 每一个 HELLO 数据包包含一个 40 字节的公钥、一个 2 字节的标识和 3 字节的随机计数器. 此外, 节点还接收或转发一个 70 字节的加密信息. 一共包含 115 字节.

因此, 每一跳的信息传送需消耗的能量为 $115\times 59.2\mu J = 6.808mJ$; 而每一跳的信息接收需消耗的能量为 $115\times 28.6\mu J = 3.289mJ$. 因此, 每一个节点平均需要消耗 40.848 mJ 和 19.734 mJ 来完成认证密钥的发送和接收操作.

在对偶密钥误用检测阶段, 为了实现检测功能, 每一个数据包额外包含一个认证因子和一个随机计数器, 它们的长度为 11 字节, 因此, 接收或发送数据包时需额外增加消耗的能量分别为 0.315 mJ 和 0.651 mJ. 值得指出的是, 在 IEEE 802.15.4 标准(该标准适用于低能量的无线传感器网络)里, 一个数据包的有效载荷可达 102 字节[18], 因此实现本算法, 仅需额外占用 10.8%的有效载荷.

5.4 计算开销分析

每一个节点的计算开销包括: 交换认证密钥、验证公钥以及验证和构造认证因子.

在椭圆曲线密码体制(ECC)里, 最耗时的计算是标量点乘(scalar point multiplication), 实验表明[17]: 在 Atmega 128L 8M 处理器上的一个 ECC－160 的标量点乘计算需要耗费 810 毫秒, 则每一个节点平均需要耗费 4860 毫秒来完成认证密钥的交换.

为了验证公钥, 每一个节点需要耗费 $k=\left\lceil \frac{m}{N}\ln 2 \right\rceil$ 个哈希操作. 每一个哈希操作在 Atmega 128L 8M 处理器需耗费 5.6 毫秒[18], 因此节点需要耗费 $\frac{3.882\times m}{N}$ 毫秒完成公钥验证工作.

为了验证认证因子, 每一节点需根据式(2)使用两个哈希操作来重构认证因子, 此外, 若对偶密钥未被误用, 节点还需使用两个哈希操作来构造新的认证因子. 因此, 在正常情况下, 节点将使用四个哈希操作来重构和新建认证因子, 这将耗费 22.4 毫秒.

若对偶密钥被误用, 节点将使用两个哈希值构造对偶密钥误用检测报告, 这将耗费 11.2 毫秒.

5.5 与典型算法分析

本算法主要与 Liu 算法进行比较. 其他针对 Liu 算法的改进[4-7]大都在降低通信开销带来的能耗, 其他安全性能方面则几乎未涉及.

1) 检测准确率

本算法由于采用了 Bloom Filter, 因此误用密钥的检测准确率几乎达到 100%, 不管有多少数量的受损节点(Bloom Filter 的误报概率是影响检测准确率的最大因素). 而 Liu 算法的检测准确率依赖于基站或当前节点所选中的检测节点, 需要可靠的通信支持, 若检测节点受损, 则无法对误用对偶密钥进行检测.

2) 检测位置

在本算法里, 每一个节点, 既是转发节点, 又是检测节点, 所有的对偶密钥检测均在一跳之内完成. 而在 Liu 算法里, 若使用基站进行检测, 则检测数据包需经过多跳才能到达; 若使用特定的检测节点进行检测, 则检测节点的部署密度必须达到相当程度才能保证每一个普通节点都能够在其通信范围之内找到至少一个检测节点, 否则将无法完成对偶密钥误用检测工作.

3) 网络拓扑结构易变的适应性

本算法无须采取其他额外的机制去处理网络拓扑结构的动态变化, 这是因为: 两个通信节点仅仅需要交换认证密钥即可实现对偶密钥误用检测, 其拓扑结构的变化不影响通信节点的判断. 而在 Liu 算法里, 若被选中的检测节点由于拓扑结构的变化而未处于当前节点的通信范围时, 不管检测节点是否受损, 对偶密钥误用检测都无法完成.

与 Liu 算法相比, 本算法的存储开销要大一些, 同时在认证密钥交换阶段带来额外的通信开销和计算开销. 但是, 对于任意一对通信节点而言, 认证密钥交换阶段仅仅被执行一次; 不存在 Liu 算法及其他算法所存在的需要多跳节点存储和通信开销的问题.

6 总结与展望

在无线传感器网络里, 一旦对偶密钥受损, 将对网络的安全通信构成威胁. 常用的节点受损检测机制只是针对受损节点的处理, 并不能够检测并发现所有的受损密钥, 因为正常节点的对偶密钥也可能受损. 因此, 从保证可靠的安全性和节约网络资源的角度来看, 对偶密钥的误用检测对于保证网络的安全通信具有重要的现实意义.

本文所提出的算法, 采用在数据包附加不可伪造和否认的认证因子的机制, 为解决在正常节点之间对

偶密钥的误用检测提供一种可行的解决方法. 该算法无需任何预先确定的节点部署信息, 也不针对特定的对偶密钥建立方式, 因此可以与其他的密钥管理算法集成, 为无线传感器网络提供更可靠、更完全的安全服务.

在本算法里, 认证密钥的生成是基于非对称密码体制, 节点需存储公/私密钥对, 而且使用 Bloom Filter 技术实现节点公钥的认证, 给节点的存储、计算带来一定的开销. 如何在不过多增加节点的开销的情况下, 为节点动态生成认证密钥是今后值得关注的研究方向.

参考文献

1 Chaddoud G. A selective survey on key distribution in sensor networks. Damascus University Journal, 2012, 28(1): 105–123

2 Liu D, Dong Q. Detecting misused keys in wireless sensor networks. International Performance Computing and Communications Conference. April 2007. 272–280.

3 Kim M, Han YJ, Park SH, Chung TM. An enhanced misused key detection mechanism in wireless sensor networks. Second International Conference on Sensor Technologies and Applications(SENSORCOMM). 2008. 645–650.

4 Park MW, Kim JM, Han YJ, Chung TM. A misused key detection mechanism for hierarchical routings in wireless sensor network. 4th International Conference on Networked Computing and Advanced Information Management(NCM). 2008. 47–52.

5 Kang DM, Park MW, Han YJ, Park SH. A misused key detection mechanism using the token in Wireless Sensor Networks. The 12th International Conference on Advanced Communication Technology (ICACT). 2010. 238–242.

6 Han YJ, Park MW, Kim JM, Chung TM. A light-weighted misused key detection in wireless sensor networks. Computational Science and Its Applications–ICCSA 2010, Lecture Notes in Computer Science Volume 6018, 2010: 352–367.

7 Goyal R, Prasad D. An efficient security scheme for wireless sensor networks. International Journal of Emerging Technology and Advanced Engineering, 2013, 3(4): 793–796.

8 Broder A, Mitzenmacher M. Network applications of bloom filters: A survey. Internet Mathematics, 2004, 1(4): 485–509.

9 Malan DJ, Welsh M, Smith MD. A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography. 1st IEEE Int' Conf. on Sensor and Ad Hoc Communications and Networks. IEEE Press. 2004. 71–80.

10 Roman R, Alcaraz C, Lopez J. A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes. Mobile, Networks and Applications (MONET) Journal, Springer, 2007, 12(4): 231–244.

11 ANSI X9.63. Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport using Elliptic Curve Cryptography. Washington, American National Standards Institute. 1998.

12 Hartung C, Balasalle J, Han R. Node compromise in sensor networks: The need for secure systems. University of Colorado Technical Report. CU-CS-990-05. December 2004.

13 Hwang J, He T, Kim Y. Detecting phantom nodes in wireless sensor networks. the 26th IEEE International Conf. on Computer Comm. (INFOCOM). 2007. 2391–2395.

14 Staddon J, Balfanz D, Durfee G. Efficient tracing of failed nodes in sensor networks. The 1st ACM International Workshop on Wireless Sensor Networks and Applications. 2002. 122–130.

15 Dong D, Liu Y, Liao X, Self-monitoring for sensor networks. the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc). 2008. 431–440.

16 Rivest R. The RC5 encryption algorithm. The 1st International Workshop on Fast Software Encryption. volume 809, 1994. 86–96.

17 Wander AS, Gura N, Eberle H, Gupta V, Shantz SC. Energy analysis of public-key cryptography for wireless sensor networks. 3rd IEEE International Conference on Pervasive Computing and Communications (PERCOM). 2005. 324–328.

18 IEEE Computer Society. IEEE 802.15.4: IEEE standard for information technology-- telecommunications and information exchange between systems local and metropolitan area networks.