Implementing the SHA-1 on the ATmega328p

Timo Selzam   
*Ingeniuerswissenschaften und Informatik*  
*TH Aschaffenburg*Aschaffenburg, Germany  
s220864@th-ab.de

*Abstract*—Durch die immer größere Verbreitung von Microcontrollern in elektronischen Schaltungen steigt die Wahrscheinlichkeit, dass diese angegriffen oder die Daten verändert werden. Um diesem Umstand zu begegnen, soll eine mathematische Einwegfunktion (SHA-1) zum Einsatz kommen. In diesem Paper präsentiere ich, wie dieser Algorithmus genutzt werden kann, um die Sicherheit bei der Nutzung von Microcontrollern zu erhöhen. Zur Präsentation wird ein ATmega328P genutzt, der die zu verschlüsselnde Nachricht, sowie die Kommandos zur Verschlüsselung und zur Ausgabe des Hashs über eine USART Schnittstelle erhält bzw. ausgibt.

Keywords—ATmega328, SHA-1, security

# Introduction

In fast allen elektronischen Geräten werden heutzutage Microcontroller zur Steuerung oder Datenverarbeitung eingesetzt. Oftmals speichern bzw. verarbeiten sie viele Daten die sie von Sensoren oder über eine externe Texteingabe erhalten. Um diese Geräte vor Manipulation oder Datendiebstahl zu schützen, soll ein Verfahren angewendet werden, um diese Gefahren zu minimieren. Geläufige Methoden, um Daten oder Passwörter zu verschlüsseln bzw. auf ihre Integrität zu überprüfen sind zum Beispiel kryptographische Hashfunktionen wie “MD5” oder “SHA 1”, “SHA 2”, “SHA 3”. Diese funktionieren alle nach einem ähnlichen Prinzip, bei dem die ursprüngliche Nachricht in Blöcke aus fest definierten Größen aufgeteilt werden. Diese werden anschließend mit vorgeschriebenen Zahlenreihen aufgefüllt und anschließend mit definierten Konstanten, Hashs und Formeln verrechnet. Selbst die kleinste Abweichung von der ursprünglichen Nachricht führt zu einem komplett anderen Ergebnis. Das zurückrechnen eines Hashs, um die original Nachricht zu erhalten ist nicht möglich, da bei der Berechnung ein Überlaufen der Variablen absichtlich in Kauf genommen wird. Aufgrund der genannten Eigenschaften solcher Funktionen soll in diesem Paper aufgezeigt werden, wie der “SHA 1” Algorithmus aufgebaut ist und wie man ihn auf einem “ATmega328p” implementieren kann.

# Theoretical questions

## Mathematical one way function

Die Definition einer mathematischen Einwegfunktion lautet: “ Eine Hashfunktion ist eine Ein-Weg-Funktion (one-way oder preimage resistant function), wenn es für einen gegebenen Fingerabdruck *z* berechnungsmäßig praktisch unmöglich ist, eine Nachricht *M* mit *h(M )* = z zu finden.” [1, p. 96]. Kurz gesagt bedeutet das, dass das Rekonstruieren der ursprünglichen Nachricht, auch mit der Kenntnis über den Hash sowie die verwendete Funktion, so schwer ist, dass diesen Umstand als praktisch unmöglich ansehen kann. Ausnahmen sollen zu einem späteren Zeitpunkt behandelt werden.

## Typical applications

Eine typische Anwendung von Einwegfunktionen stellt die Kryptographie dar. Zum Beispiel wird ein Passwort nicht in reiner Textform in einer Datenbank abgelegt, sondern es wird der Hash des Passwortes gespeichert. Auf diese Weise kann bei einer Anmeldung der berechnete Hash des eingegebenen Passworts mit dem Abgelegten überprüft werden und sollte die Datenbank kompromittiert werden, ist das vergebene Passwort nach wie vor unbekannt. Eine weitere Anwendung ist die Überprüfung auf Echtheit / Veränderung von Dateien. Wenn die ursprüngliche Datei bzw. deren Hash bekannt ist, kann die Erhaltene sehr einfach und schnell überprüft werden. Selbst eine minimale Änderung, wie zum Beispiel die Benutzung eines ‘A’ anstatt eines ‘a’ führt zu einem komplett unterschiedlichen Hash. Somit ist eine ‘sinnvolle’ Veränderung einer Datei so gut wie ausgeschlossen.

## Preimage Resitance

Der Begriff “preimage resistance” bezieht sich auf die Kollisionsfreiheit der genutzten Hashfunktion.

First Preimage Resistance: Von einer first preimage resistance spricht man, wenn es “für einen gegebenen Fingerabdruck *z* berechnungsmäßig praktisch unmöglich ist, eine Nachricht *M* mit *h(M)* = *z* zu finden”. [1, p. 96]

Second Preimage Resistance: Die second preimage resistance beschreibt, “wenn es berechnungsmäßig praktisch unmöglich ist, eine Nachricht M′ ≠ M mit h(M) = h(M′) zu finden.” [1, p. 95]

## Kollision

Von einer Kollision spricht man, wenn zwei unterschiedliche Dokumente bzw. Passwörter zu ein und demselben Hash führen. Sollte eine Hashfunktion häufig zu Kollisionen führen bzw. leicht gefunden werden können, ist dieser Algorithmus nicht als sicher anzusehen.

Wenn die Eingabe nicht beschränkt ist, wird es zwangsläufig zu Kollisionen kommen, da es mehr Eingabe- als Ausgabemöglichkeiten gibt.

## XOR-Funktion

Um die Integrität einer Nachricht zu überprüfen, ist eine reine XOR Funktion nicht geeignet, da, sobald der Schlüssel bekannt ist, jede Nachricht entschlüsselt werden kann. Des Weiteren sind Kollisionen bei einer linearen Funktion wesentlich wahrscheinlicher, als bei anderen Hash-Funktionen.

## Geburtstagsproblem

Das Geburtstagproblem beschreibt, dass ab einer Gruppengröße von 23 Personen die Wahrscheinlichkeit 50% beträgt, dass mindestens zwei Personen am gleichen Tag, unabhängig vom Jahrgang, geboren wurden. [2, p. 36] Diese Wahrscheinlichkeit lässt sich anhand folgender Formel berechnen:

mit . Für eine Wahrscheinlichkeit von 50% sei

Somit ergibt sie die Schätzung:

Die Variable sei die Anzahl der Personen und sei die Anzahl der “Nichtgeburtstage”.

Bei dem sogenannten Geburtstagsangriff werden zufällige Dateien / Nachrichten 1 bis *k* ausgewählt und der dazugehörige Hash berechnet. Danach wird verglichen, ob der gleiche Hasch zwei Mal vorkommt. Durch das Nutzen des Geburtstagsproblems werden Versuche benötigt, um mit einer Wahrscheinlichkeit von 50% eine Kollision in dem von uns genutzten SHA 1 zu finden.

Zum Vergleich: Um eine Nachricht mit dem gleichen Hash wie eine zuvor festgelegte Nachricht zu erhalten werden Wiederholungen benötigt.

Bei der Nutzung eines Rechners, der 1 Mrd. Versuche pro Sekunde schafft, benötigt dieser ca. Jahre mit dem Ausnutzen des Geburtstagsproblems und ca. Jahre ohne die Nutzung.

## Blockchain

Mathematische Einwegfunktionen spielen im Kontext von Bitcoin eine große Rolle im Hinblick auf Sicherheit, Konsens und Integrität.

Bitcoin nutzt Hashfunktionen wie zum Beispiel SHA 256 um Transaktionen sowie ganze Blöcke zu verschlüsseln. Jeder Block enthält den Hash des Vorherigen, wodurch eine nachträgliche Veränderung ausgeschlossen wird, da dies den Hash aller Nachfolgenden ungültig machen würde.

Jede Transaktion wird durch eine mathematische Einwegfunktion verschlüsselt und abschließend mit einer Signatur, erstellt durch asymmetrische Kryptographie, versehen.

# Analysis/Discussion of the Firmware

In diesem Abschnitt soll die Implementierung der Firmware beschrieben und analysiert werden. Das Hauptaugenmerk soll dabei auf die Umsetzung des SHA 1 Verfahrens, sowie auf den USART Kommandointerpreter gelegt werden.

## Implementig USART

Aufgabe der Firmware ist es, eine Zeichenkette einzulesen, zu verschlüsseln und wieder auszugeben. Um diese Funktionen ausführen zu können muss zu erst eine Möglichkeit geschaffen werden, damit Nachricht und Kommandos eingegeben werden können. In diesem Fall wird dazu die USART Schnittstelle des Atmega328p genutzt. USART steht für universal s*ynchronous* / a*synchronous* receiver transmitter.

Um eine Nachricht senden und empfangen zu können, werden die Leitungen RxD (receive), TxD (transfer) sowie ein gemeinsamer GND benötigt. Wenn die Datenübertragung synchron durchgeführt werden soll, wird noch eine weitere Leitung für das gemeinsame Taktsignal benötigt. Des Weiteren ist darauf zu achten, dass die Baudrate (Übertragungsgeschwindigkeit) von Sender und Empfänger übereinstimmt.

Damit beide Seiten die Übertragung richtig interpretieren, ist es unabdingbar das gleiche Übertragungsprotokoll zu nutzen. Dieses setzt sich aus der Nachrichtenlänge, Start- und Stoppbits und ggf. einem Paritätsbit zusammen. Die Nachrichtenlänge kann zwischen 5 und 9 Bits betragen, das Paritätsbit kann eine gerade oder ungerade Parität anzeigen oder deaktiviert werden und es besteht die Wahl zwischen einem und zwei Stoppbits.

Das Protokoll, das für die Implementierung des SHA 1 genutzt wird, nutzt eine asynchrone Übertragung, eine Datenlänge von 8 Bit, ein Stoppbit und kein Paritätsbit. Die Baudrate wird in einem anderen Programmteil festgelegt. Des Weiteren wird das Senden und Empfangen, sowie ein Empfangsinterrupt aktiviert. Die Festlegung der Parameter erfolgt in den Registern UCSR0B, UCSR0C, UBRR0H und UBRR0L. Diese Einstellungen werden vorgenommen, sobald die Funktion “usart\_init” aufgerufen wird.

Nachdem die grundlegenden Einstellungen vorgenommen wurden, werden die Fuktionen zum Senden und Empfangen der Daten geschrieben.

Die Funktion “usart\_put\_byte” wartet bis das UDRE0 Flag im Register UCSR0A durch eine Eins signalisiert, dass das UDRn (USART data register) leer und somit bereit zum übertragen ist. Diese Überprüfung ist notwendig, damit es zu keinen Kollisionen bei der Datenübertragung kommt. Danach wird das Byte, das an die Funktion übergeben wurde, in das UDRn geschrieben und somit gesendet.

An die Funktion “usart\_put\_string” wird ein Pointer bzw. Array übergeben, dass eine Zeichenkette enthält. Zuerst wird geprüft, ob an der Adresse auf die der Pointer zeigt ein Nullterminator steht. Solange dies nicht der Fall ist, wird das Byte an der aktuellen Adresse an die Funktion “usart\_put\_byte” übergeben. Anschließend wird die Adresse, auf die der Pointer zeigt um eine Variablengröße erhöht. Diese Schleife läuft solange, bis der Pointer auf eine Adresse zeigt, die den Nullterminator enthält und somit das Ende des Strings signalisiert.

Der Vollständigkeit halber wurde die Funktion “usart\_receive” ebenfalls implementiert. Sie kann genutzt werden um im Polling Verfahren aktive darauf zu warten, dass das UDRn eine Nachricht erhält. Hierzu wird das RXC0 Flag im UCSR0A Register beobachtet. Erst wenn es 1 wird, wird die while Schleife verlassen und das UDRn ausgelsen.

## Implementing SHA 1

Bevor der Code für den SHA 1 geschrieben werden kann, soll der Ablauf hier schematisch besprochen werden. Eine nähere Betrachtung findet im späteren Verlauf statt.

Der genutzte Algorithmus soll aus einer beleibigen Zeichenkette einen Hash mit 160 Bit erzeugen. Um dieses Ziel zu erreichen, wird die Nachricht in mindestens einen 512 Bit breiten Block aufgeteilt. Dieser ist wie folgt aufgebaut. Einer Nachricht mit der Länge *l* Bits wird eine einzelne “1” angehängt und mit einer Anzahl *k* Nullen aufgefüllt, wobei *k* die kleinste, nicht negative Lösung der Gleichung

sei. Dann wird ein 64 Bit Block mit der ursprünglichen Nachrichtenlänge *l* in binär Code angehängt. [3, p. 13]

Um eine fehlerfreie Funktion der Firmware sicher zu stellen ist es essentiell, dass die empfange Nachricht mit der gesendeten Nachricht übereinstimmt. Dieser Umstand spielt eine große Rolle bei der Auswahl der Baudrate, da eine höhere Übertragungsgeschwindigkeit zwar die Übertragungszeit verkürzt, aber im gleichen Zuge auch die Robustheit verringert.

Zur Bestimmung der Baudrate muss die Bytelänge (l), die maximal zu übertragenden Bytes (n) und die möglichen Baudraten (Bd) betrachtet werden. Zur Beurteilung wird die folgende Formel genutzt

|  |  |  |
| --- | --- | --- |
| Baudrate | Byte Zeit (1/8/1) | Übertragungszeit (T) |
| 1200 | 8,33 ms | 8,33 s |
| 2400 | 4,17 ms | 4,17 s |
| 4800 | 2,08 ms | 2,08 s |
| 9600 | 1,04 ms | 1,04 s |

Da, wie Eingangs schon erwähnt, die Robustheit oberste Priorität hat und eine Übertragungszeit von ca. 1,04 Sekunden ausreichend für diese Applikation ist, wählen wir eine Baudrate von 9600 Bd. Um diese später in das Register des Microcontrollers schreiben zu können, muss sie zuvor noch in das passende Format umgerechnet werden. Dazu definieren wir die Makros “F\_CPU”, “BAUDRATE” und “Teiler” und nutzen die Formel

Der Compiler wird jedes Vorkommen der Zeichenketten “F\_CPU”, ”, “BAUDRATE” und “Teiler” mit den definierten bzw. berechneten Werten ersetzen. Es ist zu beachten, dass große Werte mit UL (unsigned long) versehen werden müssen, damit diese Zahlen nicht falsch interpretiert werden.

Die Datentypen “uint32\_t” und “uint64\_t” wurden für eine bessere Lesbarkeit des Programms erstellt. Wenn der Header für einen anderen Microcontroller genutzt werden sollte ist davor sicherzustellen, dass Datentypen und längen identisch sind.

Um die Berechnung des Hashs übersichtlich gestalten zu können wurde ein Funktionspointer erstellt, auf den im späteren Verlauf eingegangen werden soll.

Die zuvor erstellten Daten “SHA\_1.h” und “usart.c” müssen in die “SHA\_1.c” per include eingebunden werden, damit sie verwendet werden können. Weiterhin werden Header zur Nutzung von Interrupts (avr/interrupt.h), zur Speicherverwaltung (stdlib.h) und zur Vereinfachung der Arbeit mit den Registern des Microcontrollers (avr/io.h) eingebunden. In “main“ werden die Funktionen “usart\_init()” und “sei()” aufgerufen um die USART Schnittstelle zu initialisieren und externe interrupts zu aktivieren. Die pointer “M” und “W” werden erzeugt. Diese werden im späteren Verlauf genutzt um die eingegebene Nachricht in 512 Bit Blöcke aufzuteilen bzw. die für den SHA 1 Algorithmus benötigten 80 Rechenschritte durchzuführen. Damit nicht unnötig Speicher reserviert werden muss erfolgt die Speicherallokation dynamisch bei der Berechnung des Hashs, da erst zu diesem Zeitpunkt abschließend bekannt ist, wie viel Speicher tatsächlich benötigt wird. Der Speicher des Buffers “msg”, der bereits in der “SHA\_1.h” erzeugt wurde, wird über die Funktion “malloc()” mit der Breite von 8 Bit erzeugt und einer Adresse zugewiesen.

Damit das Programm nicht nur einmal durchlaufen und danach beendet wird, findet die Abarbeitung in einer while Schleife statt. Die Bedingung der Schleife wird mit “1” angegeben, damit sie immer erfüllt ist.

Um den Speicher das Atmega328p nicht zu überstrapazieren wurde die maximale Nachrichtenlänge auf 1000 Zeichen festgelegt. In jedem Durchlauf des Programms wird geprüft, ob diese Restriktion eingehalten wurde. Ist dies nicht der Fall, wird eine Fehlerroutine aufgerufen und der Programmablauf unterbrochen.

Über eine Verbindung zu einem PC kann die Nachricht über einen seriellen Monitor an den Atmega328p geschickt werden. In dem Programmcode des Microcontrollers wird die eingehende Nachricht über eine ISR (interrupt service routine) in den Puffer “msg” geschrieben. Bei jedem Aufruf der ISR wird der Speicher des Puffers um zwei weitere Bytes durch die Funktion “realloc” erweitert. Dies ist nötig um das neu eingegangen Byte der Nachricht ablegen und den Nullterminator anhängen zu können. Sollte die Speicherreallokation schief gehen springt das Programm in einer Fehlerroutine und eine weitere Abarbeitung wird unterbrochen. Bei einer erfolgreichen Reallokation wird das empfangene Byte im Puffer abgelegt und der Längenindex “len” um eins erhöht. Es können solange Daten eingegeben werden bist entweder die Nachrichtenlänge 1000 überschreitet oder bis das Kommando zur Berechnung des Hash eingegeben wird.

Die Eingabe des Kommandos wird mit einer if Abfrage fortlaufend kontrolliert. Wenn erkannt wird, dass das vorletzte Zeichen “#” und das letzte Zeichen “!” ist, wird die Verarbeitung gestartet. Aufgrund der Vorgaben ist es somit nicht möglich die Zeichen “#” und “!” hintereinander in der zu verschlüsselnden Nachricht zu nutzen.

Im ersten Schritt der Bearbeitungsrutine wird die benötigte Blockanzahl ermittelt. Hierzu wird der Umstand genutzt, dass bei einer Division von integer Variablen die Nachkommastellen entfallen. In diesem speziellen Fall wird der Längenindex der Nachricht mit 8 Multipliziert und das Ergebnis durch 448 geteilt. Anschließend wird das eine eins addiert, um die Anzahl *n* der benötigten 512 Bit Blöcke zu erhalten. Die Berechnung in “Bits” wurde gewählt um eine einheitliche Datengrundlage zu schaffen.

Nach der Berechnung werden die letzten zwei Stellen des Puffers, die nur das Kommando darstellen, mit dem Nullterminator beschrieben.

Im weiteren Verlauf wird mit der Funktion “calloc” Speicher für “M” und “W” allokiert. Das Besondere an der genutzten Funktion ist, dass der allokierte Speicher gleich zu Beginn mit “0” beschrieben wird. Somit muss der Block nicht mit *k* Nullen aufgefüllt werden. “W” stellt die einzelnen 512 Bit Blöcke dar ()und “M” wird zur Berechnung der Durchläufe genutzt. Auch hier wird überprüft, ob die Speicherallokation erfolgreich war. Falls nicht wird eine Fehlerroutine aufgerufen und der Programmablauf gesperrt.

Bei dem Aufruf der Funktion *pad()* werden die Pointer *msg* und *W* übergeben. Hierbei handelt es sich um einen *call by reference* Aufruf, der es ermöglicht, die Puffer in der Funktion zu bearbeiten. Die Funktion *pad()* wird genutzt um die einzelnen Bytes des Puffers *msg* in die Wortorganisierten Blöcke *M* zu schreiben, nach dem letzten Zeichen der Nachricht eine 1 anzuhängen. Außerdem wird die ursprüngliche Nachrichtenlänge in Bit in den letzten beiden Wörtern des letzten Blocks geschrieben.

Zur Umsetzung werden zwei *if* Schleifen genutzt. Die erste Schleife wird zur Indizierung der 16 *n* Worte W genutzt. In der Zweiten wird bei jedem Durtchgang mittels der zuvor erstellten temporären Zählvariable *count* geprüft, ob der Puffer *msg[count]* den Nullterminator enthält. Wenn dies nicht der Fall ist wird das enthaltene Zeichen mittels der Zählvariablen *k* der Schleife und einer mathematischen Berechnung auf die vorgesehene Position geschoben. Damit bei dem Schiebevorgang keine Daten verloren gehen muss ein Typecast des 8 Bit breiten Zeichens auf einen 32 Bit breiten Datentypen erfolgen. Anschließend wird die Zählvariable  *count* um 1 inkrementiert.

Wenn die Zeichenkette aus *msg* vollständig übertragen wurde wird über *count* berechnet wo die Nachricht endet und eine einzelne 1 an diese Position angehängt. Es wird eine temporäre Variable *bits* angelegt, in der die Länge *l* der ursprünglichen Nachricht in Bits gespeichert wird. Das vorletzte Wort des letzten Blocks wird, mit Zuhilfenahme des Schiebeoperators rechts, mit den oberen 32 Bit der Variable beschrieben, das letzte Wort des letzten Blocks wird mit den unteren 32 Bit der Variable beschrieben.

Das Hauptaugenmerk wurde auf die zur Berechnung des Hashs benötigte Funktion *process()* gelegt, da sie die komplexeste des ganzen Programms ist. Sie sollte follumfänglich nutzbar sein, was bedeutet, dass auch Zeichenkette mit mehr als 55 Symbolen eingegeben & mehrere Hashs hintereinander erzeugt werden können. Außerdem sollte sie übersichtlich und ressourcenschonen sein. Um diesen Anforderungen zu erfüllen wird ein Funktionspointer genutzt. Der Vorteil dabei ist es, dass der Aufruf der zum Berechnungsvorgang benötigten Formel über eine Indizierung erfolgt. Somit wird nicht jede Formel zur Berechnung in eine eigene Schleife eingebettet sondern der Index zur passende Formel anhand des aktuellen Berechnungsschritts berechnet. Die vorgegebenen Konstanten wie auch die initialen Hashs wurden jeweils in einem Array abgelegt und können somit auch mit dem berechneten Index ausgewählt werden.

Aufgrund der genutzten Pointer, Arrays und Funktionen ist es möglich die Berechnung mit zwei *for* Schleifen durchzuführen und den Code dafür schmal, überschaubar und gut lesbar zu gestalten.

Zu Beginn wird die temproäre Variable *temp* erstellt. Diese wird benötigt um einen berechneten Wert zwischen zu speichern und zu einem späteren Zeitpunkt in einen der fünf Puffer (*A* bis *E)* zu speichern. Für den Fall, dass die Berechnung mit mehr als einem 512 Bit breiten Block stattfinden soll wird die erste Schleife benötigt. In ihr wird zu Beginn der aktuellste Hashwert in die Puffer *A* bis *E* gespeichert. Bei dem ersten Durchlauf entspricht dieser dem Initialwert.

Für die 80 Berechnungsdurchläufe wird die zweite Schleife und für die Indizierung die Zählvariable *t* genutzt. Während der ersten 16 Durchläufe gilt

wobei *i* die Zählvariable der ersten Schleife ist. Anhand der Berechnung des Indexes kann man erkennen, dass mit jedem weiteren Durchlauf der ersten Schleife der Index des Puffers *M* um *i* 16 erhöht wird. Nach dem 16. Schritt wird W[t] wie folgt berechnet

Die Funktion *rotl(x, y)* führt eine zyklische Verschiebung des übergebenen Wertes *x* um die Anzahl *y* Bits nach links durch. Der Unterschied zu einer reinen Schiebeoparation liegt darin, dass die herausgeschobenen Bits nicht verloren gehen, sondern von rechts wieder eingeschoben werden.

Zur Berechnung der Variablen *temp* wird folgende Formel verwendet

Sie enthält, neben den bereits bekannten Puffern *W* und  *E* und Funktionen, das Array *K* in dem die Konstanten gespeichert sind und den Funktionspointer *funktion*. Für die Indizierung der Beiden wird die Zählvariable *t* der zweiten Schleife durch den Wert 20 dividiert. Durch diese einfache Berechnung ist es möglich alle 80 Durchgänge in nur einer Schleife zu bearbeiten. Anschließend werden die Puffer *A* bis *E* in jedem Durchgang neu beschrieben.

Nachdem die Berechnung abgeschlossen ist werden die Puffer *A* bis *E* zu den Hashs *H[0]* bis *H[4]* aufaddiert. Nachdem alles 512 Bit breiten Blöcke berechnet wurden enthalten die Hashpuffer *H[0]* bis *H[4]* die finalen Werte.

Nach der Berechnung muss der Speicher der Puffer *M* und *W* wieder freigegeben werden, da durch die Berechnung eines neuen Hashs wieder Speicher allokiert wird. Dies passiert, bis kein Speicher mehr vorhanden ist, was zu unvorhersehbaren Auswirkungen führen kann. Für die Freigabe des Speichers wird in der Funktion *de\_init()* die Funktion *free* genutzt und der Pointer auf den NULL Vektor gerichtet.

Damit der Programmfluss gesteuert werden kann wird die Variable *failsafe* auf 1 gesetzt.

## Units

* Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as “3.5-inch disk drive”.
* Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
* Do not mix complete spellings and abbreviations of units: “Wb/m2” or “webers per square meter”, not “webers/m2”. Spell out units when they appear in text: “. . . a few henries”, not “. . . a few H”.

Identify applicable funding agency here. If none, delete this text box.

* Use a zero before decimal points: “0.25”, not “.25”. Use “cm3”, not “cc”. (*bullet list*)

## Equations

The equations are an exception to the prescribed specifications of this template. You will need to determine whether or not your equation should be typed using either the Times New Roman or the Symbol font (please no other font). To create multileveled equations, it may be necessary to treat the equation as a graphic and insert it into the text after your paper is styled.

Number equations consecutively. Equation numbers, within parentheses, are to position flush right, as in (1), using a right tab stop. To make your equations more compact, you may use the solidus ( / ), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

*a**b* 

Note that the equation is centered using a center tab stop. Be sure that the symbols in your equation have been defined before or immediately following the equation. Use “(1)”, not “Eq. (1)” or “equation (1)”, except at the beginning of a sentence: “Equation (1) is . . .”

## Some Common Mistakes

* The word “data” is plural, not singular.
* The subscript for the permeability of vacuum **0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter “o”.
* In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
* A graph within a graph is an “inset”, not an “insert”. The word alternatively is preferred to the word “alternately” (unless you really mean something that alternates).
* Do not use the word “essentially” to mean “approximately” or “effectively”.
* In your paper title, if the words “that uses” can accurately replace the word “using”, capitalize the “u”; if not, keep using lower-cased.
* Be aware of the different meanings of the homophones “affect” and “effect”, “complement” and “compliment”, “discreet” and “discrete”, “principal” and “principle”.
* Do not confuse “imply” and “infer”.
* The prefix “non” is not a word; it should be joined to the word it modifies, usually without a hyphen.
* There is no period after the “et” in the Latin abbreviation “et al.”.
* The abbreviation “i.e.” means “that is”, and the abbreviation “e.g.” means “for example”.

An excellent style manual for science writers is [7].

# Using the Template

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.

## Authors and Affiliations

**The template is designed for, but not limited to, six authors.** A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

### For papers with more than six authors: Add author names horizontally, moving to a third row if needed for more than 8 authors.

### For papers with less than six authors: To change the default, adjust the template as follows.

#### Selection: Highlight all author and affiliation lines.

#### Change number of columns: Select the Columns icon from the MS Word Standard toolbar and then select the correct number of columns from the selection palette.

#### Deletion: Delete the author and affiliation lines for the extra authors.

## Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is “Heading 5”. Use “figure caption” for your Figure captions, and “table head” for your table title. Run-in heads, such as “Abstract”, will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced. Styles named “Heading 1”, “Heading 2”, “Heading 3”, and “Heading 4” are prescribed.

## Figures and Tables

#### Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 1”, even at the beginning of a sentence.

1. Table Type Styles

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| copy | More table copya |  |  |

1. Sample of a Table footnote. (*Table footnote*)
2. Example of a figure caption. (*figure caption*)

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

##### Acknowledgment *(Heading 5)*

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

##### References

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

1. G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*
2. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
3. I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
4. K. Elissa, “Title of paper if known,” unpublished.
5. R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.
6. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
7. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

**IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper not being published**